Издательство «Знание»

ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА И ЕЁ ПРИМЕНЕНИЕ

Новое в жизни, науке, технике

Подписная научно-популярная серия

Издается ежемесячно с 1988 г.

Машинные коды

1992

3

Новое
в жизни,
науке,
технике

ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА И ЕЁ ПРИМЕНЕНИЕ

Подписная
научно-
популярная
серия

3/1992

Издается
ежемесячно
с 1988 г.

МАШИННЫЕ КОДЫ

В номере:

В.С.Жданов, С.Л.Окунев, В.Ю.Горелов
КОМАНДЫ МИКРОПРОЦЕССОРОВ К580 И МИКРО-ЭВМ «ЭЛЕКТРОНИКА»

А.В.Чумак
СИСТЕМНЫЕ ЯЧЕЙКИ ПЭВМ-0010

Ю.В.Котов
СОВМЕСТНОЕ ИСПОЛЬЗОВАНИЕ БЕЙСИКА И МАШИННЫХ КОДОВ

В.Ф.Очков, В.А.Васин
СУБД ИРИС

И.П.Березенцев
ЗАГРУЗКА КОДОВЫХ ПРОГРАММ В КУВТ-86

Издательство
«Знание»
Москва
1992

ББК 32.973
М 38

Авторы выпуска

В.С.ЖДАНОВ, доктор технических наук, С.Л.ОКУНЕВ, кандидат технических наук, В.Ю.ГОРЕЛОВ — специалисты Московского института электронного машиностроения, кафедра «Вычислительные системы и сети»

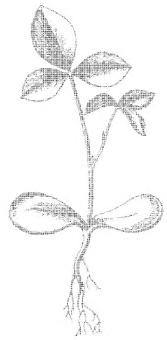
А.В.ЧУМАК, программист

Ю.В.КОТОВ, заведующий лабораторией МАДИ, программист, разработчик версий АВГРАФ

В.Ф.ОЧКОВ, кандидат технических наук, доцент, программист (МЭИ), В.А.ВАСИН, программист

И.П.БЕРЕЗЕНЦЕВ, инженер-программист

РЕДАКТОР Б.М.ВАСИЛЬЕВ



Scan AAW

ISBN 5-07-002498-1

© Издательство «Знание», 1992 г.

В первом номере сборника за 1989 год была опубликована статья «Программирование на языке АССЕМБЛЕР». По просьбе читателей ниже приводится список команд микро-ЭВМ «Электроника 60» (реализуемый в микропроцессорах К1801ВМ1, ВМ2) и микропроцессора КР580ВМ80 с необходимыми пояснениями. Материал статьи — описание машинных команд распространенных процессоров с соответствующими пояснениями — необходим для пользователей ПЭВМ и интересен для многих читателей.

В.С.ЖДАНОВ, С.А.ОКУНЕВ, В.Ю.ГОРЕЛОВ

Команды микропроцессоров К580 и микро-ЭВМ «Электроника»

Система команд МП КР580ВМ80А

В МП КР580ВМ80 используется 11 форматов команд (рис.1), коды операций (КОП) которых имеют различную длину (2, 5, 6 или 8 бит). На рис.1 приняты следующие обозначения:

КОП — код операции;

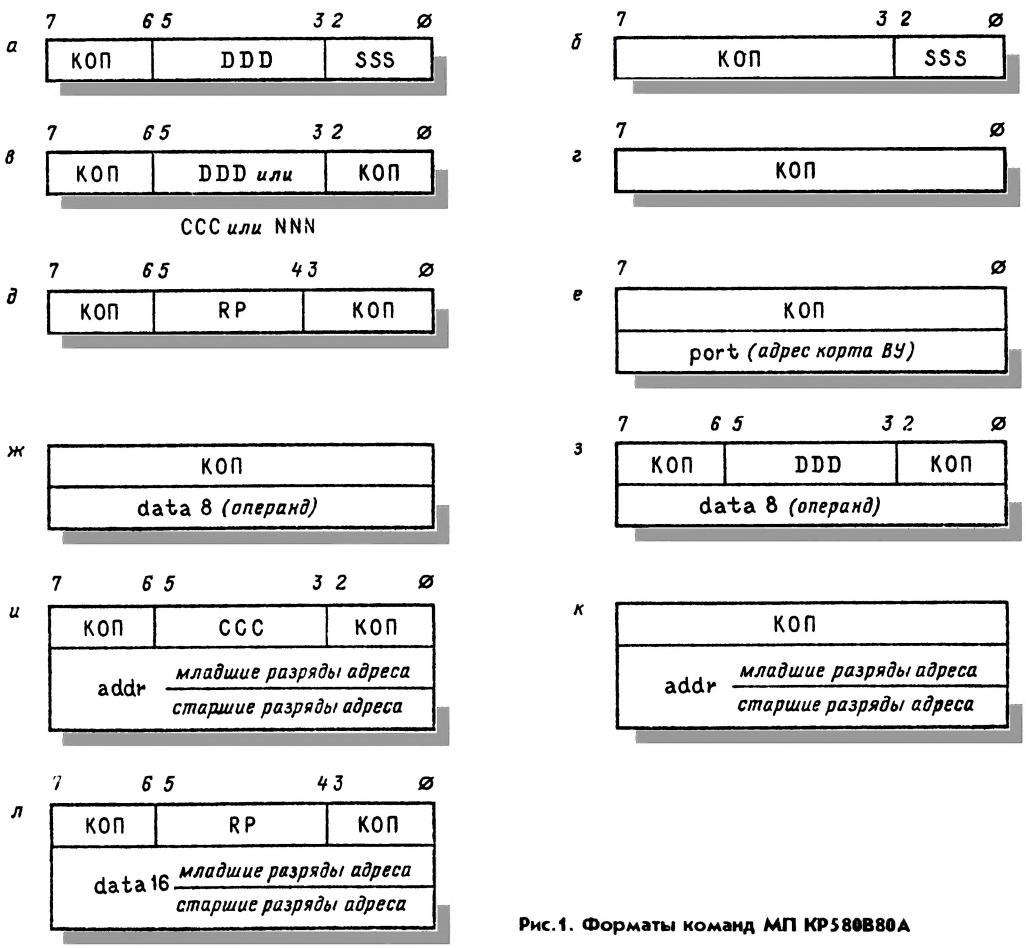


Рис.1. Форматы команд МП КР580ВМ80А

DDD и SSS — трехбитовые поля кода приемника и источника информации;

RP — двухбитовое поле кода используемой регистровой пары;

port — адрес порта внешнего устройства;

data 8 — 8-битовый операнд;

addr — 2-байтовый адрес;

data 16 — 16-битовый операнд;

CCC — трехбитовое поле кода условия, используемого в команде;

NNN — число от 1 до 7, номер повторного запуска МП.

В зависимости от способа адресации команды могут быть одно-, двух- или трехбайтовыми. В командах используются четыре способа адресации:

1) регистровая (MOV R1,R2; ADD R; PCHL;...);

2) косвенно-регистровая (MOV M,R; ADD M; PUSH; POP;...);

3) непосредственная (MVI R, data 8; ADI data 8;...);

4) прямая LDA addr, SHLD addr; IN port;...).

Трехбитовые поля адресации источника и приемника информации обозначаются в машинных командах символами SSS и DDD соответственно, а двухбитовое поле адресации регистровой пары — символами RP. Коды регистров и регистровых пар представлены на рис.2. В командах условного перехода, условного обращения к подпрограмме и условного возврата из подпрограммы используются коды условий (CCC) (рис.3).

Команды МП КР580ВМ80А приведены в табл. 1-3, на которых:

1) в мнемонических изображениях двухадресных команд приемник указывается на первом месте, а источник — на втором;

2) формат каждой команды указан на рис.4-6 в виде ссылки на соответствующий формат рис.1;

Таблица 1

Список команд пересылки МП КР580В80А

Мнемоника	Двоичный код	Формат	Число тактов	Название и описание
				Пересылка
MOV R1,R2	01DDDSSS	1.а	5	(R2) –> (R1)
MOV R,M	01DDD110	1.а	7	(M) –> (R)
MOV M,R	01110SSS	1.а	7	(R) –> (M)
MVI R, data 8	00DDD110	1.в	7	data –>(R)
MVI M, data 8	00110110	1.в	10	data –> (M)
				Загрузка
LXI RP, data 16	00RP00101	1.л	10	data –> (RP)
LDA addr	00111010	1.к	13	(addr) –> (A)
LHLD addr	00101010	1.к	16	(addr) –> (L), (addr+1) –> (H)
LDAX RP	00RP1010	1.д	7	(RP) –> (A)
				Обмен
XCHG	11101011	1.г	4	(H)–>(D), (L)–>(E)
				Запись
STA addr	00110010	1.к	13	(A) –> (addr)
SHLD addr	00100010	1.к	16	(L) –> (addr) (H) –> (addr+1)
STAX RP	00RP0010	1.д	7	(A) –> (RP)

Таблица 2

Мнемоника	Двоичный код	Формат	Число тактов	Название и описание	Устанавливаемые признаки
				Сложение	
ADD R	10000SSS	1.б	4	(A)+(R) –> (A)	S, Z, P,
ADD M	10000110	1.б	7	(A)+(M) –> (A)	C, AC
ADI data 8	11000110	1.ж	7	(A)+data 8 –>(A)	
				Сложение с переносом	
ADC R	10001SSS	1.б	4	(A)+(R)+(C) –> (A)	S, Z, P,
ADC M	10001110	1.б	7	(A)+(M)+(C) –> (A)	C, AC
ACI.data 8	11001110	1.ж	7	(A)+data 8+(C)–>(A)	
				Вычитание	
SUB R	10010SSS	1.б	4	(A)-(R) –> (A)	S, Z, P,
SUB M	10010110	1.б	7	(A)-(M) –> (A)	C, AC
SUI data 8	11010110	1.ж	7	(A)-data 8 –> (A)	
				Вычитание с заемом	
SBB R	10011SSS	1.б	4	(A)-(R)-(C) –> (A)	S, Z, P,
SBB M	10011110	1.б	7	(A)-(M)-(C) –> (A)	C,AC
SUI data 8	11011110	1.ж	7	(A)-data 8-(C)–>(A)	
				Логическое умножение	
ANA R	10100SSS	1.б	4	(A) AND (R) –> (A)	S, Z, P,
ANA M	10100110	1.б	7	(A) AND (M) –> (A)	C, AC
ANI data 8	11100110	1.ж	7	(A) AND data 8–>(a)	
				Исключение ИЛИ	
XRA R	10101SSS	1.б	4	(A) XOR (R) –>A	S, Z, P,
XRA M	10101110	1.б	7	(A) XOR (M) –> (A)	C ,AC
XRI data 8	11101110	1.ж	7	(A) XOR data 8 –>(A)	
				ИЛИ	
ORA R	10110SSS	1.б	4	(A) OR (R) –> (A)	S, Z, P,
ORA M	10110110	1.б	7	(A) OR (M) –> (A)	C, AC
ORI data 8	11110110	1.ж	7	(A) OR data 8 –> (A)	
				Сравнение	
CMP R	10111SSS	1.б	4	(A) – (R)	S, Z, P,
CMP M	10111110	1.б	7	(A) – (M)	C = 0
CPI data 8	11111110	1.ж	7	(A) – data 8	AC = 0
				Увеличение	
INR R	00DDD000	1.в	5	(R)+1 –> (R)	S, Z, P,
INR M	00110100	1.в	10	(M)+1 –> (M)	AC
INX RP	00RP0011	1.д	5	(RP)+1 –> (RP)	–
				Уменьшение	
DCR R	00DDD101	1.в	5	(R)-1 –> (R)	S, Z, P,
DCR M	00110100	1.в	10	(M)-1 –> (M)	AC
DCX RP	00RP1011	1.д	5	(RP)-1 –> (RP)	–

Продолжение табл. 2

Мнемоника	Двоичный код	Формат	Число тактов	Название и описание	Устанавливаемые признаки
DAD RP				**Сложение содержимых регистровых пар**	
RLC	OORP1001	1.д	10	(H,L)+(RP) –> (H,L)	C
				Циклический сдвиг влево	
	00000111	1.г	4	Все биты A смещаются на один разряд влево. Старший разряд переходит в его нулевой разряд и регистр признака	C
RRC				**Циклический сдвиг вправо**	
	00001111	1.г	4	Выполняется аналогично RLC. Старшие разряды A и C приобретают значение младшего разряда A	C
RAL				**Циклический сдвиг влево через перенос**	
	00010111	1.г	4	Все биты A сдвигаются на один разряд влево; старший разряд A переходит в C, а C – в младший разряд A	C
RAR				**Циклический сдвиг вправо через перенос**	
	00011111	1.г	4	Выполняется аналогично RAL,но сдвиг вправо	C
DAA				**Десятичная коррекция аккумулятора**	
	00110111	1.г	4	Преобразование содержимого A в двоично-десятичный код	S, Z, P, C, AC
CMA				**Инвертирование A**	
	00101111	1.г	4	Получение обратного кода (инверсии) содержимого A	–
SIC				**Установка C**	
	00110111	1.г	4	1 –> C	C = 1
CMC				**Инвертирование C**	
	00111111	1.г	4	Инвертирование содержимого регистра C	C

Таблица 3

Список команд управления, ввода-вывода и работы со стеком

Мнемоника	Двоичный код	Формат	Число тактов	Название и описание
JMP addr	11000111	1.к	10	**Безусловный переход** к команде, адрес которой определяется вторым и третьим байтами команды
PCHL	11101001	1.г	5	**Косвенный переход** по адресу, указанному в регистрах H и L
Jcc addr	11CCC110	1.и	5/10	**Условные переходы:** если условие (cc из рис.3) истинно, то переход к команде, адрес которой определяется вторым и третьим байтами команды; иначе выполняется команда, расположенная вслед за данной
CALL addr	11001101	1.к	17	**Обращение к подпрограмме:** содержимое счетчика команд заносится в стек по адресу, на который указывает SP; содержимое SP уменьшается на 2, и выполняется переход к команде, адрес которой определяется вторым и третьим байтами команды
Ccc addr	11CCC100	1.и	11/17	**Условное обращение к подпрограмме:** если условие (cc из рис.3) истинно, то выполняется команда CALL; иначе выполняется команда, расположенная вслед за данной
RET	11001001	1.г	10	**Возврат из подпрограммы:** переход к команде, адрес которой записан в верхней паре ячеек стека и увеличение содержимого SP на 2
Rcc	11CCC000	1.в	5/11	**Условный возврат из подпрограммы:** если условие (cc из рис.3) истинно, то выполняется команда RET; иначе выполняется команда, расположенная вслед за данной
RST n	11NNN111	1.в	11	**Повторный запуск МП** с адреса 8-NNN (0,8,16,...56)
PUSH RP	11RP0101	1.д	11	**Загрузка стека** содержимым пары регистров (RP)
POP RP	11RP0001	1.д	10	**Выдача данных из стека** в пару регистров (RP); при выдаче PSW происходит установка S,Z,P,C и AC
XTHL	11100011	1.г	18	**Обмен** содержимым верхней пары ячеек стека и пары регистров H,L
SPHL	11111001	1.г	5	**Пересылка** содержимого пары регистров H,L в указатель стека SP
IN port	11011011	1.е	10	**Ввод** в аккумулятор данных из адресуемого порта
OUT port	11010011	1.е	10	**Вывод** данных из акумулятора в адресуемый порт
EI	11111011	1.г	4	**Разрешение прерывания**
DI	11110011	1.г	4	**Запрещение прерывания**
HLT	01110110	1.г	7	**Останов**
NOP	00000000	1.г	4	**Нет операции**

Регистры		Пары регистров			
Код (SSS или DDD)	Имя (R, R_1 или R_2)	Код (RP)	Имя пары (RP)	Регистры пары	
				старший	младший
000	B	00	B	B	C
001	C				
010	D	01	D	D	E
011	E				
100	H	10	H	H	L
101	L				
110	M (память)	11	PSW	A	F
111	A(аккумулятор)				

Рис. 2. Коды регистров и пар регистров, используемые в командах МП

Код (ccc)	Мнемоника (cc)	Условие (значение разряда регистра F)
000	NZ	Не нуль (Z = 0)
001	Z	Нуль (Z = 1)
010	NC	Нет переноса (C = 0)
011	C	Перенос (C = 1)
100	PO	Нечетность (P = 0)
101	PE	Четность (P = 1)
110	P	Плюс (S = 0)
111	M	Минус (S = 1)

Рис. 3. Коды условий, используемые в кодах условных переходов

3) признаки результата (S,Z,P,C и AC) устанавливаются лишь при выполнении большинства арифметических и логических команд, а также команды POP PSW;

4) мнемонические обозначения команд условного перехода, условного обращения к подпрограмме и условного возврата из подпрограммы составляются из символов J, C и R и соответствующих символов (CC) из рис.3 (например, JNC, CP, RM);

5) число тактов, необходимых для исполнения команд условного обращения к подпрограмме и условного возврата из подпрограммы, зависит от того, выполнено (знаменатель дроби) или не выполнено (числитель дроби) условие, указанное в команде;

6) в описаниях команд для обозначения содержимого регистра, аккумулятора или ячейки памяти, на которые указывают SSS или DDD, используется запись (R), (A) или (M);

7) при записи мнемоники команды RP обозначает имя регистровой пары, а при записи кода команды — двухбайтовый код регистровой пары (см. рис.2).

Система команд микро-ЭВМ «Электроника 60»

В микроЭВМ «Электроника 60" используются шесть основных форматов команд (рис.4), коды операций которых имеют различную длину. В зависимости от режима адресации командами с форматами рис. 4,б,г могут использовать дополнительное слово памяти, а команды с форматами рис.4,а — даже два дополнительных слова памяти (рис. 4,ж,з).

На рис.4 приняты следующие обозначения:

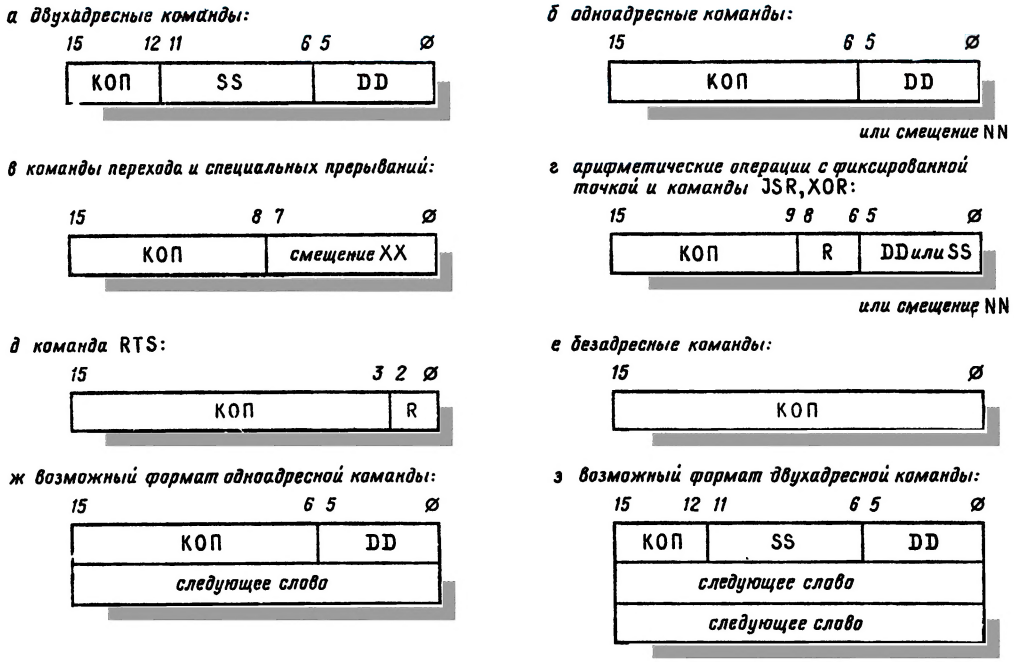


Рис.4. Основные форматы команд микро-ЭВМ «Электроника 60»

КОП — код операции;

SS и DD — шестибитовые поля, определяющие источник и приемник информации;

XX — восьмибитовое поле команд перехода, определяющее смещение;

NN — шестибитовое поле, определяющее смещение в командах MARK и SOB;

R — номер (от 0 до 7) регистра общего назначения.

Операнд (операнды) в команде задается 6-разрядным полем, включающим 3-разрядный номер регистра (N = 0,1,2,...,7) и 3-разрядный указатель режима адресации (рис.5).

В системе команд микроЭВМ «Электроника 60" имеется 12 способов адресации:

1) **Регистровая адресация** (режим 0). Операндом является содержимое регистра (формат RN, где N — число от 0 до 7).

2) **Косвенная регистровая адресация** (режим 1). Исполнительный адрес операнда определяется содержимым регистра (формат @RN или (RN)).

3) **Автоинкрементная адресация** (режим 2). Исполнительный адрес операнда определяется содержимым регистра, затем содержимое регистра увеличивается на длину операнда в байтах (1 или 2) (формат (RN)+).

4) **Косвенная автоинкрементная адресация** (режим 3). Косвенный адрес определяется содержимым регистра, затем содержимое регистра увеличивается на 2. Исполнительный адрес выбирается из ячейки памяти по косвенному адресу (формат @(RN)+).

5) **Автодекрементная адресация** (режим 4). Содержимое регистра уменьшается на длину операнда в байтах (1 или 2), затем исполнительный адрес операнда выбирается из регистра (формат -(RN)).

6) **Косвенная автодекрементная адресация** (режим 5). Содержимое регистра уменьшается на 2, затем косвенный адрес выбирается из регистра, а исполнительный адрес — из ячейки памяти по косвенному адресу (формат @-(RN).

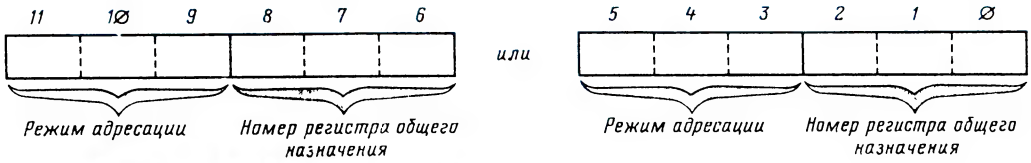


Рис.5. Указание операнда в командах формата

7) **Индексная адресация** (режим 6). Содержимое регистра (базы) и очередного слова в потоке команд (смещение) суммируются для получения исполнительного адреса, а счетчик команд будет указывать на слово, расположенное вслед за смещением (формат E(RN), где E — выражение, определяющее величину смещения). В этом (а также следующем) режиме адресации команда должна иметь одно или два дополнительных слова. Если команда двухадресная и для каждого из операндов задан косвенный режим адресации, то в слове, следующем за словом команды, должно размещаться смещение для вычисления адреса первого операнда (SS), а в следующем слове — смещение для вычисления адреса второго операнда (DD) (рис. 4,з).

8) **Косвенная индексная адресации** (режим 7). Содержимое регистра (база) и очередного слова в потоке команд (смещение) суммируются для получения косвенного адреса; исполнительный адрес выбирается из ячейки памяти по косвенному адресу (формат @E(RN)).

Режимы адресации 2, 3, и 7 очень часто применяются с использованием регистра R7 — счетчика команд. Поэтому эти случаи выделены в отдельные способы адресации.

9) **Непосредственная адресация** (режим 2, регистр 7). Операнд содержится в очередном слове команды (формат #E). Содержимое счетчика команд (регистр R7) автоматически увеличивается на 2 после выборки из памяти каждого нового слова, что обусловливает после выборки операнда переход к следующему а ним слову в потоке команд.

10) **Абсолютная адресация** (режим 3, регистр 7). Исполнительный адрес содержится в очередном слове команды (формат @#E, где E — выражение, задающее исполнительный адрес операнды).

11) **Относительная адресация** (режим 6, регистр 7). Считывается и временно запоминается очередное слово команды (смещение); содержимое счетчика команд увеличивается на 2, что обусловливает пропуск смещения и переход к следующему за ним слову в потоке команд. Исполнительный адрес получается путем суммирования смещения (сохраняемого слова) с новым содержанием счетчика команд (форма E, где E — выражение, задающее смещение).

Адресация называется относительной потому, что адрес операнда вычисляется относительно текущего значения счетчика команд; смещение — это расстояние (в байтах) между операндом и словом, на которое указывает счетчик команд в момент вычисления адреса операнда. Если команда и операнд перемещаются в памяти так, что расстояние между ними остается постоянным (равным смещению), то команда будет выполняться правильно в любом месте памяти.

12) **Косвенно-относительная адресация** (режим 7, регистр 7). Очередное слово команды (смещение) суммируется с новым содержимым счетчика команд; в результате формируется адрес адреса, т.е. косвенный адрес операнда (формат @E).

Основные команды микроЭВМ «Электроника 60» приведены в табл. 4-6, в которых:

1) в командах, которые могут оперировать со словами и байтами, приводятся два мнемонических обозначения (байтовые команды отмечаются дополнительным символом B) и соответствующие им восьмеричные коды;

2) шестибитовые поля адресации источника и приемника информации кодируются в командах символами SS и DD соответственно, трехбитовые поля регистров — символом R, шестибитовые смещения — NN, а восьмибитовые смещения — XX;

3) в описаниях команд для обозначения содержимого регистра или слова памяти, на которые указывают SS и DD, используется запись (SS) или (DD);

4) формат команды указан в виде ссылки на соответствующий формат рис.4;

5) коды условий (N, Z, V, C) устанавливаются по результату выполнения

Таблица 4

Список двух- и одноадресных команд микро-ЭВМ «Электроника 60»

Мнемоника	Восьмеричный код	Формат	Название и описание	Коды условий после выполнения команды
			Группа двухадресных команд	
MOV MOVB	01SSDD 11SSDD	4.а	**Пересылка.** Содержимое SS пересылается в DD	N=1, если (SS) <0 Z=1, если (SS) <0 V=0; C=const
CMP CMPB0	2SSDD 12SSDD	4.а	**Сравнение.** Установка кодов условий в соответствии с результатом вычитания содержимого DD из содержимого SS. Результат вычитания никуда не записывается, DD и SS не изменяются	N=1, если RES < 0 Z=1, если RES = 0 V=1, если возникло арифм. переполнение C=1, если был заем
BIT 03SSDD	BITB 13SSDD	4.а	**Проверка битов.** Установка кодов условий в соответствии с результатом логической операции И над содержимыми SS и DD (без записи результата)	N=1, если ст. разряд RES равен 1 Z=1, если RES = 0 V=0; C = const
BIC BICB	04SSDD 14SSDD	4.а	**Очистка битов.** Обращает в 0 те разряды (DD), которым соответствуют установленные в 1 разряды (SS)	N=1, если ст. разряд RES=1 Z=1, если RES = 0 V=0, C = const
ADD	06SSDD	4.а	**Сложение.** Содержимое SS складывается с содержимым DD и результат пересылается в DD	N=1, если RES < 0 Z=1, если RES = 0 V=1, если возникло арифм.переполнение C=1, если был перенос
SUB	16SSDD	4.а	**Вычитание.** Содержимое SS вычитается из содержимого DD, и результат пересылается в DD	N=1, если RES < 0 Z=1, если RES = 0 V=1, если возникло арифм. переполнение C=1, если был заем
MUL	070RSS	4.г	**Умножение.** Содержимое SS умножается на содержимое R, и результат (два слова) пересылается в R и следующий за ним регистр	N=1, если RES < 0 Z=1, если RES = 0 V=0 C=1, если RES $<-2^{15}$ или RES $>2^{15}-1$
DIV	071RSS	4.г	**Деление.** Содержимое R и следующего за ним регистра делится на содержимое SS. Частное помещается в R, а остаток — в следующий за ним регистр	N=1, если RES < 0 Z=1, если RES = 0 V=1, если (SS) = 0 C=1, если (SS) = 0

Продолжение табл. 4

Мнемоника	Восьмеричный код	Формат	Название и описание	Коды условий после выполнения команды
XOR	074RSS	4.г	**Исключающее ИЛИ.** Над содержимым R и SS выполняется операция исключающее ИЛИ	N=1, если RES < 0 Z=1, если RES = 0 V=0, C = const
			Группа одноадресных команд	
CLR CLRB	0050DD 1050DD	4.б	**Очистка.** Обращает в 0 содержимое DD	N=V=C=0 Z=1
COM COMB	0051DD 1051DD	4.б	Инвертирование. Получение обратного кода (инверсии) содержимого DD (результат в DD)	N=1, если ст. разряд RES=1; Z=1, если RES=O; V=0, C=1
INC INCB	0052DD 1051DD	4.б	**Увеличение.** Содержимое DD увеличивается на 1 (результат в DD)	N=1, если RES < 0 Z=1, если RES = 0 V=1, если возникло арифм. переполнение
DEC DECB	0053DD 1053DD	4.б	**Уменьшение.** Содержимое DD уменьшается на 1 (результат в DD)	N=1, если REC < 0 Z=1, если RES = 0 V=1, если (DD) = 10000000 ИЛИ 200 (при работе с байтами); C = const
NEG NEGB	0054DD 1054DD	4.б	**Изменение знака.** Содержимому DD присваивается знак «минус» (содержимое DD заменяется на свое двоичное дополнение)	N=1, если RES < 0 Z=1, если RES = 0 V=1, если (DD) = 100000 или 200 (при работе с байтами) C=0, если RES = 0
ADC ADCB	0055DD 1055DD	4.б	**Сложение с переносом.** Содержимое разряда C регистра состояния складывается с содержимым DD (результат в DD)	N=1, если RES < 0 Z=1, если RES = 0 V=1, если возникло арифм. переполнение; C=1, если был заем
SBC SBCB	0056DD 1056DD	4.б	**Вычитание переноса.** Содержимое разряда C регистра состояний вычитается из содержимого DD (результат в DD)	N=1, если RES < 0 Z=1, если RES = 0 V=1, если возникло арифм.переполнение; C=1, если был заем
TST TSTB	0057DD 1057DD	4.б	**Проверка.** Установка кодов условий в соответствии с содержимым DD	N=1, если DD < 0 Z=1, если DD = 0 V=C=0
ROR RORB	0060DD 1060DD	4.б	**Циклический сдвиг вправо.** Все биты содержимого DD смещаются на один разряд вправо. Младший разряд DD переходит в C, а C — в старший разряд DD	N=1, если RES < 0 Z=1, если RES = 0 V=N XOR C C загружается младшим разрядом DD
ROL ROLB	0061DD 1061DD	4.б	**Циклический сдвиг влево.** Работает аналогично ROR, но сдвиг влево	N, Z, V — см. ROR. C загружается старшим разрядом DD

Продолжение табл. 4

Мнемоника	Восьмеричный код	Формат	Название и описание	Коды условий после выполнения команды
ASR ASRB	0062DD 1062DD	4.б	**Арифметический сдвиг вправо.** Все биты DD, кроме старшего, смещаются на 1 разряд вправо. Младший бит DD переходит в C	N=1, если RES < 0 Z=1, если ES = 0 V, N XOR C C загружается младшим разрядом DD
ASL ASLB	0063DD 1063DD	4.б	**Арифметический сдвиг влево.** Все биты DD, кроме старшего, смещаются на 1 разряд влево. В младший разряд DD записывается 0, а C заполняется старшим разрядом DD	N=1, если RES < 0 Z=1, если RES = 0 V = N XOR C C загружается старшим разрядом DD
SWAB	0003DD	4.б	**Обмен (свопинг) байтов.** В операнде DD перестановка байтов. Результат оценивается по младшему байту	N=1, если 7-й разряд DD=1 Z=1, если младший байт = 0 V=C=0
SXT	0067DD	4.б	**Расширение знака.** Содержимое DD заполняется нулями или единицами в зависимости от значения N (0 или 1)	Z=1, если (DD) = 0 N = V = C = const

Таблица 5

Список команд управления программой микро-ЭВМ «Электроника 60»

Мнемоника	Восьмеричный код	Формат	Название и описание
JMP	0001DD	4.б	Переход. Передает управление по адресу, определяемому операндом DD
BR	001XX	4.в	Безусловный переход
BNE	002XX	4.в	Переход, если не равно нулю (Z=0)
BEQ	003XX	4.в	Переход, если равно нулю (Z=1)
BGE	004XX	4.в	Переход, если больше или равно нулю (N XOR V=0)
BLT	005XX	4.в	Переход, если меньше нуля (N XOR V=1)
BGT	006XX	4.в	Переход, если больше нуля (Z OR(N XOR V)=0))
BLE	007XX	4.в	Переход, если меньше или равно нулю (Z OR(N XOR V)=1))
BPL	200XX	4.в	Переход по полюсу (N=0)
BMI	201XX	4.в	Переход по минусу (N=1)
BHI	202XX	4.в	Переход, если больше (S OR Z=0)
BLOS	203XX	4.в	Переход, если меньше или равно (C OR Z=1)
BVC	204XX	4.в	Переход, если нет переполнения (V=0)
BVS	205XX	4.в	Переход, если есть переполнение (V=0)
BCC	206XX	4.в	Переход, если нет переноса (C=0)
BCS	207XX	4.в	Переход, если есть перенос (C=1)

Продолжение табл. 5

Мнемоника	Восьмеричный код	Формат	Название и описание
SOB	077RNN	4.г	Вычитание и переход. При каждом выполнении SOB содержимое R уменьшается на 1, и если (R)≠0, то выполняется возврат на NN слов. При (R)=0 осуществляется следующая за SOB команда программы
JSR	004RDD	4.г	Обращение к подпрограмме. Содержимое R заносится в стек, в R записывается адрес возврата (содержимое счетчика команд), а в счетчик команд заносится DD (адрес первой команды подпрограммы). Подпрограмма должна сохранять значение R, чтобы можно было выполнить возврат в вызывающую программу по команде RTS R
RTS	00020R	4.д	Возврат из подпрограммы. Содержимое R (адрес возврата) пересылается в счетчик команд, и восстанавливается (извлекается из стека) прежнее значение R
RTI	000002	4.е	Возврат из подпрограммы обработки прерывания
MARK	0064NN	4.б	Возврат из подпрограммы с восстановлением указателя стека. Стек освобождается от NN параметров, которые были занесены в него перед обращением к подпрограмме, и затем работает так же, как и RTS

Таблица 6

Список служебных команд микро-ЭВМ «Электроника 60»

Мнемоника	Восьмеричный код	Формат	Название и описание
CLC	000241	4.е	Очистка C (C=0)
CLV	000242	4.е	Очистка V (V=0)
CLZ	000244	4.е	Очистка Z (Z=0)
CLN	000250	4.е	Очистка N (N = 0)
SEC	000261	4.е	Установка C (C=1)
SEV	000262	4.е	Установка V (V=1)
SEZ	000264	4.е	Установка Z (Z=1)
SEN	000270	4.е	Установка N (N=1)
CCC	000257	4.е	Очистка признаков (C=V=Z=N=0)
SCC	00277	4.е	Установка признаков (C=V=Z=N=1)
NOP	000240	4.е	Нет операции
HALT	000000	4.е	Основ работы процессора и прекращение передач по общей шине
RESET	000005	4.е	Сброс ВУ — установка ВУ в исходное состояние
WAIT	000001	4.е	Ожидание прерывания
MFPS	1067DD	4.б	Чтение содержимого регистра состояния
MTPS	1064DD	4.б	Запись содержимого регистра состояния

операции (RES), значению операнда и (или) типу операции. При этом в командах, которые могут оперировать со словами и байтами, в качестве результата (операнда) берется соответственно слово или один из байтов этого слова;

6) все числа в табл.4-6 восьмеричны;

7) команды условного перехода и команда BR позволяют осуществить передачу управления на расстояние, ограниченное смещением XX, интерпретируемым машиной как целое число со знаком в дополнительном коде: от (10000000) = – 128 до (01111111) = + 127 . В случае выполнения условия это число удваивается машиной, его 7-й бит копируется в старших разрядах (с 8-го по 15-й) и полученное 16-разрядное число прибавляется к содержимому счетчика команд;

8) в команде перехода JMP и обращения к подпрограмме JSR допускается использовать только те способы адресации, при которых в качестве операнда задается адрес ячейки памяти.

Для составления команды на языке Ассемблер микроЭВМ «Электроника 60" необходимо с помощью мнемоники команды записать действия, которые требуется выполнить, а затем, выбрав способ адресации операнда (или операндов), описать местонахождение операнда (см. форматы способов адресации).

Например:

1) требуется обнулить содержимое ячейки памяти, адрес которой находится в регистре R2. Наши действия:

а) выбираем команду очистки из рис.7 CLR, а ее код 0050DD;

б) к данной ячейке мы можем обратиться с помощью косвенного способа адресации с использованием регистра R2: (R2), код адресации – 12.

Теперь соединим эти две части команды и получим:

КОП	DD	мнемоника команды	формат способа адресации
0050	2	CLR	(R2) – команда очистки ячейки памяти, адрес которой записан в R2
восьмеричнй код команды		мнемоника Ассемблера	

2)Переслать содержимое ячейки памяти по адресу 100 в регистр R3:

а) команда пересылки MOV – код 01SSDD;

б) обратиться к ячейке памяти (операнду – источнику информации) можно с помощью абсолютной адресации: @#100 код: 27;

в) обратиться к регистру R3 (операнд=приемник) можно с помощью регистровой адресации:

R3	код 03
	код номера используемого режима регистра

Соединив код команд (КОП), операнд=источник (SS) и операнд=приемник (DD), получим:

КОП	SS	DD		
01	27	03	– 1 слово	MOV @#100, R3
00	01	00	– 2 слово	
машинный код				мнемоника Ассемблера

Приведенный материал носит справочный характер. Ограниченный объем статьи не позволяет рассмотреть вопросы, связанные с системой прерываний, особенности реализации которой зависят не только от типа микропроцессора, но и от способа построения вычислительной машины в целом. Авторы готовы предложить вниманию читателей материалы, посвященные этим и другим вопросам, связанным с программированием МП-систем на языке Ассемблер.

Литература

1. Майоров С.А., Кириллов В.В., Приблуда А.А. Введение в микроЭВМ. – М.: Машиностроение, 1988.

2. Вигдорчик Г.В., Воробьев А.Ю., Проценко В.Д. Основы программирования на Ассемблере для СМ ЭВМ. – М.: Финансы и статистика, 1983.

3. Жданов В.С., Горелов В.Ю., Окунев С.Л, Саксонов Е.А. Программирование на языке АССЕМБЛЕР. Микропроцессоры. – М.: Знание, 1989. – (Новое в жизни, науке, технике. Сер. «Вычислительная техника и ее применение»; № 1).

Статья содержит сведения о ячейках системной области ПЭВМ БК-0010 и БК-0010.01, а также данные о некоторых полезных подпрограммах, зашитых в ПЗУ монитора БК. Эти сведения будут весьма полезны для пользователей, программирующих на Ассемблере и в машинных кодах.

А.В.ЧУМАК

Системные ячейки ПЭВМ БК-0010

Для ПЭВМ БК-0010 имеется стандартная система ввода-вывода, поддерживаемая программами, записанными в ПЗУ по адресам **100000** — **120000**, организующая работу ТВ-монитора, клавиатуры, кассетного магнитофона, а также простейшую связь по локальной сети (одностороннюю передачу данных).

Так как большинство программ из-за недостатка памяти в большей или меньшей степени используют программы, зашитые в ПЗУ, то более полная информация о них выгодна, так как приводит к сокращению их объема (особенно в системе Бейсик).

Но так как они ориентированы на определенную цель — поддержание драйверов, то их применение специфично, и можно указать лишь несколько подпрограмм, которые имеют общее применение:

102062 — Звук с возрастающей частотой. Задержки при звучании: R2 — длительность (R2 не более 300_8), R3 — частота;

100472 — восьмеричный ввод. Результат ввода в R5;

100460 — печать упакованной строки. Адрес строки в R3;

100644 — печать текста «ОШИБКА»;

110346 и **110362** — сохранение/восстановление регистров R0 — R3 в стеке (регистр R4 служит для связи, вызов командой JSR R4,... , после вызова R4 не использовать);

110376 — очистка экрана;

102566 — прокрутка экрана на одну строку вниз;

105232 — очистка строки справа от курсора;

103600 — сдвижка в строке;

103700 — раздвижка в строке;

103760 — сдвижка в тексте (удаление одной строки);

104020 — раздвижка в тексте (вставка пустой строки).

Эти подпрограммы, реализующие функции управляющих кодов экрана, легко модифицируются для составления собственных программ, работающих с экраном.

Больший интерес может представлять использование программ ПЗУ по своему назначению, с расширением их функций через использование системных ячеек программ ПЗУ, расположенных по адресам **000** — **377**.

Системные ячейки ПЭВМ БК-0010

Адреса и значения пишутся в восьмеричном виде, снабжены краткими комментариями.

000 — 002:

Эти два слова стандартными подпрограммами не используются, их назначение у машин типа PDP-11 неизвестно (они зарезервированы для дальнейшего использования).

Для БК-0010 эти ячейки могут использоваться для обработки ненужных прерываний, например:

А) Обнуляем ненужные вектора прерываний.

Б) Записываем в ячейках 000-002 команду JMR <на обработку ошибочных прерываний>, или проще RTI для обработки ненужных прерываний (типа TRAP или сбоя питания .

Данный способ позволяет экономить память при частой замене адресов обработки прерываний.

004 — 006:

Эти два слова занимают PC и PSW прерывания по ошибке передачи данных. Прерывание вызывается нажатием клавиши «СТОП» или ошибкой при передаче данных (ошибка возникает при попытке процессора записать слово состояния процессора по адресу 160002, так как по этим адресам располагается ПЗУ тестов), или командой процессора HALT, или же прерывание вызывают команды, нарушающие четность PC и SP.

010 — 012:

Эти два слова занимают PC и PSW прерывания по псевдокоманде, т.е. по неправильным командам ЦП или коду, не соответствующему ни одной из команд, это:

7 – 77	75000 – 76777
210 – 237	106500 – 106677
7000 – 7777	107000 – 107777
70000 – 73777	170000 – 177777

Неправильные команды:

JMP R0 – JMP R7

JSR R0,... – JSR R7,...

Псевдокоманды можно широко использовать при эмуляции, как это было сделано в программе ПИФ (ВТ № 5, 1989).

014 — 016:

Эти два слова занимают PC и PSW прерывания по биту T – trace. Установка этого бита позволяет отслеживать фрагменты программ. Установку этого бита лучше осуществлять так:

А) Установить вектор прерывания по биту T на отслеживающую программу.

Б) Заслать в стек слово 20, затем адрес начала отслеживаемого фрагмента.

В) Выполнить команду RTI.

```
TRACE$: MOV #A$,@#14
MOV #200,@#16
MOV #ADR-BEG, – (SP)
MOV #20, – (SP)
RTI
```

Отслеживающая программа (VIEW$) должна выполнять действия по просмотру необходимых регистров или ячеек памяти:

```
A$: JSR PC,VIEW$
CMP (SP), #ADR-END
BEQ RET$
RTI
RET$: CMP (SP)+,(SP)+RET
```

020 — 022:

Эти два слова занимают PS и PSW прерывания по команде IOT. Команда может использоваться для эмуляции.

024 — 026:

Эти два слова занимают PC и PSW прерывания по аварии питания. Рекомендуется установить на команду RTI – это повысит надежность вашей программы (возможна ситуация, когда сигнал на линии ACLO возникнет, а достаточно значительных сбоев не произойдет, и ваша программа останется работоспособной).

030 — 032:

Эти два слова занимают PC и PSW прерывания по команде EMT. Диспетчер EMT команд расположен с адреса 100112, а таблица адресов подпрограмм, вызываемых командой EMT с 100004-100110. По адресу 100002 находится слово 254, т.е. команда EMT 2 передает управление подпрограмме по адресу 254 и может использоваться пользователем. Также могут использоваться команды EMT с аргументами, большими 110, для вызова различных подпрограмм. (Такое

возможно из-за отсутствия контроля за аргументами ЕМТ команд в диспетчере, но надо быть осторожным, например, команда ЕМТ 0 пытается перейти по адресу 167, вызывая тем самым прерывание по нечетному адресу — вектор прерывания 4). Вот список этих команд, вызывающих подпрограммы под стеком или в области ПЗУ:

ЕМТ 130 и 150 — 100000 (Переинициализация системы);
ЕМТ 156 — 100442 (Инициализация клавиатуры и выход в монитор);
ЕМТ 272 — 120000 (Выход в систему БЭЙСИК или ФОКАЛ).
ЕМТ 254 — 0
ЕМТ 116 — 2
ЕМТ 144 — 20
ЕМТ 200 — 120
ЕМТ 172 — 200
ЕМТ 324 — 240
ЕМТ 356 — 750
ЕМТ 262 — 1000
ЕМТ 370 — 1002
ЕМТ 370 — 1002
ЕМТ 364 — 1006

034 — 036:

Эти два слова занимают PC и PSW прерывания по команде TRAP. Прерывание TRAP стандартными драйверами не используется, в системах ФОКАЛ и БЕЙСИК используется в служебных целях: для вывода символов с кодами 0-177 и для вывода сообщения об ошибке. Также можно широко использовать для эмуляции и т.д.

Флаги (байты) состояния дисплея:

040:
Флаг режима 32/64 символа в строке.
041:
Флаг инверсии фона общей зоны экрана.
042:
Флаг режима расширенного ОЗУ.
043:
Флаг РУС/ЛАТ. Кроме того, этот байт служит маской для формирования кодов русских и латинских букв из кодов клавиатуры. Стандартное значение этого байта 200, поэтому при записи в него кода 240 можно будет вводить с клавиатуры только коды заглавных русских букв, кода 40 — только заглавных латинских букв, а код 377 вовсе блокирует ввод букв — клавиатура выдает символ «ъ» (клавиши ЗАГЛ и СТР формируют 5-й бит кода клавиатуры).
044:
Флаг режима ПОДЧеркивание символа.
045:
Флаг режима ИНВерсии символа.
046:
Флаг режима ИНДикации Символов Управления.
047:
Флаг режима БЛокировки РЕДактирования.
050:
Флаг ГРАФического режима.
051:
Флаг режима ЗАПоминания.
052:
Флаг режима СТИРания. При одновременной установке флагов СТИР и ЗАП работает режим ЗАП. Флаг ГРАФ служит флагом включения одного из графических режимов.
053:
Флаг режима 32/64 символов в служебной строке.
054:
Флаг режима ПОДЧеркивания символов в служебной строке.
055:

Флаг режима ИНВерсии символов в служебной строке.

056:

Флаг режима погашенного курсора.

Ненулевое значение флага свидетельствует о включенном режиме. Отмена режимов осуществляется побитной инверсией флага — командой COM FLAG (кроме флага РУС/ЛАТ), и поэтому при содержимом байта, не равном 0 и 377, режим невозможно выключить, т.е. команда INC FLAG фиксирует данный режим, если он не был включен, и выключает, если режим был включен. Команда DEC FLAG включает режим, если он не был включен, и фиксирует в обратном случае.

060 — 062:

Эти два слова содержат PC и PSW прерывания по нижнему регистру клавиатуры. Адрес обработки — 101136.

064 — 076:

Эти шесть слов не используются драйверами системного ПЗУ.

100 — 102:

Эти два слова содержат PC и SP пользовательского прерывания IRQ2 по сигналу, поданному на контакт ПРТ порта ввода-вывода. Драйверами, БЕЙСИКОМ, ФОКАЛОМ не используются.

При инициализации вектор 100 обнуляется.

БЕЙСИК устанавливает на команду RTI. Адрес — 135070.

ФОКАЛ устанавливает на остаток своей функции FCLK (от ДВК-й версии). Адрес — 132226.

Драйвер клавиатуры:

104:

Код последней нажатой клавиши:

105:

Флаг несчитанного кода, содержит последний несчитанный код. При чтении обнуляется (для EMT 6). До тех пор пока значение этой линейки не равно 0, нажатие всех прочих клавиш игнорируется.

106:

Слово задержки при ожидании отжатия клавиши «ПОВТ».

110:

Флаг нажатия клавиши «ПОВТ».

111:

Флаг и число выдаваемых пробелов функцией EMT 6 (элемент табуляции). Ненулевое значение байта указывает на то, что при обращении к EMT 6 функция выдает код пробела.

112 — 120:

32/64 битовое поле. Каждый установленный бит — затабулированная позиция экрана. Для режима 32 символа в строке слова 116 и 120 не используются.

122:

Слово содержит число выдаваемых символов строки ключа. При нулевом значении строка ключа не выдается. Возможное применение этой ячейки — для определения дополнительных ключей клавиатуры, которые вызываются через нажатие клавиши СУ и какой-нибудь буквенной клавишей из набора @,A-G.

Программа, реализующая эту функцию, перехватывает прерывание по верхнему регистру клавиатуры, и при наличии кода ключа устанавливает ячейку 122. Адреса строк ключей располагаются по адресам 126 — 150 (строки упакованные). Вот текст этой программы в формате системы MIRAGE:

```
001000) 004437 110340 JSR R4,@#110340
001004) 010546 MOV R5,-(SP)
001006) 004737 102052 CALL @#102052
001012) 120027 000007 CMPB RO,#7
001016) 101402 BLOS 1024
001020 000137 101150 JMP @#101150
```

001024) 105737 000122 TSTB @#122
001030) 001375 BNE 1020
001032) 006300 ASL RO
001034) 062700 000102 ADD #102,RO
001040) 000137 101450 JMR @#101450

Программа перемещаема, для ее работы необходимо заслать по адресу 60 адрес ее начальной точки. Слово 102 в команде ADD #102,R0 — это смещение к стандартному расположению массива адресов строк ключей.

124:

Адрес очередного печатаемого символа строки ключей.

126 — 150:

Эти десять слов занимает массив адресов упакованных строк ключей.

Ячейки драйвера дисплея

152:

Флаг нормализации положения курсора. Подпрограмма, осуществляющая нормализацию, располагается по адресу 102414 и осуществляет коррекцию координат курсора при выходе из рабочего поля, а также если установлен флаг 152, осуществляет привязку X,Y координат курсора (ячейку 156) к ячейке 160.

154:

Маска для формирования точек графики, рисуемой курсором в графическом режиме. Возможное применение аналогично ячейке 155.

155:

Маска для формирования точек графики. В режиме 64 символа в строке используется 1 младший бит, при 32 символах в строке — 2 бита. Если же установить остальные биты, то точки графики будут формироваться из большего числа битов, т.е. возможно одну точку формировать из одного байта, проводить линии 3-й — 8-й толщины.

156:

Это слово содержит позицию курсора в формате X+100*У (для 64 символов в строке) и X*2+100*У (для 32 символов в строке).

160:

Это слово содержит адрес верхнего левого (при режиме 32 в строке) угла изображения курсора. При печати символа сначала (для формирования его изображения) используется эта ячейка, а не ячейка 156, а лишь потом осуществляется привязка ячейки 156 к 160-й, поэтому лучше для установки курсора изменять ячейку 160, в противном случае пользоваться нормализацией курсора.

162:

Это слово занимает смещение курсора после печати символа. При режимах 32/64 содержимое ее — 2 или 1, но его можно изменить. Чтобы курсор после печати символа смещался в другом направлении, надо заслать в ячейку 162 такие значения:

Режим 64 символа в строке:
1 — вправо;
177777 — влево;
100 — вниз;
177700 — вверх.

Режим 32 символа в строке:
2 — вправо;
177776 — влево;
100 — вниз;
177700 — вверх.

Значение более 1-2 дает печать символов вразрядку.

Данную функцию смещения мог бы выполнять байт, но из-за отсутствия в системе команд команды байтового сложения используется слово.

164:

Это слово используется при нормализации курсора, в подпрограммах перевода режимов, служит ограничением сверху для ячейки 156. Устанавливая ячейку 164 и другие ячейки, можно перейти на другой размер рабочей области экрана. При обычном режиме равна 3000, в режиме РП — 400.

166:

Это слово используется как расстояние от начала рабочего поля до курсора в режиме графики.

170:

Содержит адрес байта, в котором находится центр графического курсора. В обычном режиме 42000, в режиме РП—72000 (аналогична ячейке 160).

172:

Длина строки, ограничивающей рабочее поле курсора в режиме графики, обычно обнулена, используется в подпрограммах команды EMT 20.

174:

Это слово содержит X,У координаты курсора в режиме графики (аналогично ячейке 156).

176:

Это слово занимает X координата графического курсора.

200:

Это слово занимает У координата графического курсора.

202:

Это слово содержит адрес начала поля служебной строки. Для обычного режима вывода 40000, для режима РП-70000.

204:

Это слово содержит ширину поля служебной строки, т.е. величину, которую надо прибавить к содержимому ячейки 202 для получения адреса начала рабочей области экрана. Обычно равна 2000. При нулевом значении служебная строка не изображается, то есть ячейка 204 — флаг.

206:

Это слово содержит размер области видеоОЗУ в байтах, обычно 40000, в режиме РП-10000.

210:

Это слово содержит размер рабочей области экрана, т.е. без учета служебной строки, обычно 36000, т.е. 360 (240 десятич.) телевизионных строк, в режиме РП — 6000 и 60(48).

Устанавливая ячейки 164, 170, 202, 204, 206, 210, можно добиться необходимого размера рабочей области экрана, для того чтобы использовать программы или данные, хранящиеся в видеоОЗУ (так, например, поступает система MIRAGE).

212:

Это слово содержит маску для формирования фона. Используется при очистке экрана, формировании символов и т.д.

214:

Это слово содержит маску для формирования изображения графики, символов в режиме 32 символа в строке, подчеркивания и т.д.

216:

Это слово содержит маску для формирования фона служебной строки. Используется аналогично ячейке 212:

220:

Это слово содержит маску для формирования линии служебной строки. Очистив эту ячейку и нормализовав служебную строку кодом 236, можно погасить линию служебной строки. Это слово также служит флагом для заполнения кодом 30 или 10020 (в зависимости от режима 32/64) одной телевизионной строки ниже линии (100 байт).

222:

Это слово инкриминируется при печати символа, обнуляется при нажатии клавиши и последующей печати символа, т.е. равно (<число напечатанных при нажатии клавиши символов> — 1).

224:

При нажатии клавиши происходит прерывание от клавиатуры, которое обрабатывается по адресам 101136 или 101362. При обработке вызывается подпрограмма подзвучки по адресу 102032, которая в зависимости от флага 224 вызывает подпрограмму нормализации режимов служебной строки по адресу 110536, которая проверяет текущие режимы дисплея и устанавливает соответствующие индикаторы в служебной строке. В этой подпрограмме ячейка 224 служит счетчиком. Ячейку 224 устанавливают коды, изменяющие режимы дисплея. Во многих программах нормализация служебной строки является нежелательной, поэтому после выдачи управляющих кодов лучше очистить ячейку 224 во избежание нормализации при случайном нажатии клавиш.

226 — 252:

Эти слова в стандартных подпрограммах не используются, но их использование возможно на БК, оснащенных локальной сетью, другими устройствами.

254:

Это слово является задержкой при передаче на ТЛГ-линию. Ее содержимое в зависимости от режима:

Номер	Скорость (бод)	Задержка
1	4800	36
2	2400	103
3	1200	213
4	600	435
5	300	1100
6	150	2206
7	75	4422
10	50	6633

256:

Это слово не используется подпрограммами обслуживания порта ввода/вывода за неимением таковых, но в руководстве системного программиста сказано о желательности сохранения выводимой информации в порт в ячейке 256.

260:

Это слово содержит адрес обработки программами пользователя нажатия клавиши. При нулевом содержимом обработки не происходит, т.е. ячейка 260 также является флагом.

262:

Это слово является флагом передачи вместо кода 12 (ПС) кода 15 (ВК). Эта возможность практически не используется программами на БК, поэтому по каким-то причинам можно блокировать передачу кода 12 (например, для защиты).

264:

Адрес последнего загруженного в ОЗУ с магнитофона или линии массива данных.

266:

Длина последнего загруженного в ОЗУ с магнитофона или линии массива данных.

270 — 272:

Эти два слова не используются.

274 — 276:

Эти два слова содержат PC и PSW прерывания по нижнему регистру клавиатуры. Стандартный адрес обработки 101362.

300:

Подпрограммами чтения с магнитофона используются как флаг чтения в инверсном сигнале.

301:

Байт ответа драйвера магнитофона.

302:

Это слово используется как флаг фиктивного чтения.

304:

Это слово используется как смещение к счетчику при чтении байта данных, поэтому можно организовать чтение массивов наоборот, вразрядку.

306:

Используется для сохранения адреса блока параметров.

310:

Используется для сохранения SP.

312:

Контрольная сумма массива.

314:

Это слово используется как средняя длина сигнала при чтении для контроля. Вычисляется как средняя длина 200 сигналов настроечной последовательности, умноженная на 1,5. Возможное применение в качестве отношения скорости ЦП и магнитофона.

316:

Не используется.

Блок параметров магнитофона

Расположение блока с адреса 320 рекомендуется в Руководстве и используется пусковым монитором. В Руководстве сказано: «При недостатке места для системного стека драйвера (наибольшая глубина 16) надо расположить блок параметров с другого адреса». Однако при работе драйвера в качестве стека не используется никакая другая область памяти, SP не изменяется, ячейка 310 служит для восстановления стека после прерывания «СТОП» для доступа к данным (вектора прерывания 4) и его восстановления.

320:

Байт команды.

321:

Байт ответа.

322:

Слово адреса массива на запись и чтение.

324:

Слово длины массива на запись и чтение.

326 — 345:

Имя массива на запись или чтение (20 (16 десятич.) символов).

346:

Слово адреса текущего массива.

350:

Слово длины текущего массива.

352:

Имя текущего массива (20(16) символов).

372 — 376:

Эти три слова не используются.

При работе с локальной сетью используются программы, зашитые в ПЗУ Фокала или Бэйсика.

Фокал:

Вектор прерывания 360 (канал связи 11). Адрес обработки 136730.

Бэйсик:

Вектор прерывания 320 (канал связи 7). Адрес обработки 146404.

Список литературы:

Математическое обеспечение микро-ЭВМ «Электроника БК-0010"
Руководство системного программиста. — Т.1. — Кн. 2.

Заключение рецензента

Примечания к тексту:

1. Для сохранения и последующего восстановления содержимого регистров R0 – R5 можно рекомендовать следующие последовательности команд Ассемблера:

```
MOV R5,-(R6)
MOV R4,-(R6)   – сохранение в стеке
JSR R4,@#110346
и
JSR R R4,@#110362
MOV(R6)+,R4    – восстановление из стека
MOV(R6)+,R5
```

2. Различие между командами RTI и RTT.

При установленном в «1» четвертом бите («Т») слова состояния процессора (ССП) после выполнения каждой очередной команды происходит прерывание по вектору с адресом 14_8. При выходе из подпрограммы обработки прерывания по команде RTI бит «Т» ССП сбрасывается в 0, а по команде RTT – устанавливается в 1, поэтому программа, приведенная в качестве примера использования прерывания по биту Т, должна заканчиваться командой RTT, а не RTI:

```
A$:  JSR PC, VIEW$
CMP (SP), #ARD-END
BEQ RET$
RTT
.....
```

и т.д.

3. В «Вычислительной технике» № 11 за 1991 год приведен более полный список «нестандартных» ЕМТ-прерываний.

4. В вычислительной технике № 12 за 1992 год имеется более подробное описание флагов состояния дисплея, использование ячейки цвета фона, а также подпрограмм сохранения значений регистров в стеке и ввода восьмеричного числа в R5.

5. При значениях 100_8 и 177700_8 в ячейке 162_8 происходит сдвиг курсора после печати каждого очередного символа на одну графическую (а не текстовую) строку вверх или вниз.

6. Ячейка 172_8 содержит количество точек смещения графического курсора, указанное цифровыми клавишами в режиме ГРАФ.

Д.Ю.УСЕНКОВ

РОССИЙСКАЯ АКАДЕМИЯ НАУК
НАУЧНО-ПРОИЗВОДСТВЕННОЕ ПРЕДПРИЯТИЕ

ГЕОКОМ ЛТД

Высылаем готовые дополнительные блоки для БК0010. БК0010.01:

- ОЗУ-128 (8 страниц по 16Кб, в адр. 120000-157777 или 140000-177777, в корпусе, подключаемое к разъему МПИ, с питанием от БК);
- ЭОЗУ-8 (энергонезависимое ОЗУ, 8Кб, адр. 120000 или 140000);
- ИЗМЕРИТЕЛЬНУЮ ПРИСТАВКУ (АЦП, ЦАП, таймер – измерение напряжения, сопротивления, температуры, времени, осциллограф и т.д.);
- КОНТРОЛЛЕР НГМД (аналог КНГМД от БК0011).

Высылаем техническую документацию (схемы, чертежи, описания) на различные оригинальные приставки к БК (20 комплектов).

КАТАЛОГ С ОПИСАНИЯМИ И ЦЕНАМИ ВЫСЫЛАЕТСЯ БЕСПЛАТНО

Адрес: 683024 Петропавловск-Камчатский, а/я 150 . Рассказову Е.

Пользователи ПВМ БК-0010 с языками Фокал и Бейсик знают об относительных достоинствах и недостатках этих машин, об ошибках в программном обеспечении, «зашитом» в ПЗУ. Особо существенным ограничением БК с Бейсиком является малый объем оперативной памяти в сочетании с крайне неэкономной в отношении расхода памяти системой обработки Бейсика. Естественно поэтому обращение к программированию на Ассемблере непосредственно в машинных кодах или к смеси Бейсика и машинных программ (подпрограмм).

КОТОВ Ю.В.

Совместное использование Бейсика и машинных кодов

Машинные команды микропроцессора БК-0010 за отдельными исключениями те же, что у машин СМ, ДВК, PDP, и среди советов владельцам БК предлагают пользоваться Ассемблерами этих машин как таковыми или переделанными для БК версиями. Полностью покрыть задачи программирования такие Ассемблеры, однако, не могут, так как микропроцессор не имеет даже команд умножения и деления целых чисел, не говоря об операциях с нормализованными ("плавающими") числами и о вычислении функций. Такие операции выполняются по подпрограммам, но загружать эти подпрограммы (позаимствовав, допустим, у машины ДВК) в оперативную память неэффективно опять же в связи с ограниченностью ОП. Целесообразнее для всех возможных операций и функций использовать программные блоки и подпрограммы интерпретатора Фокала либо системы обработки Бейсика, которые зашиты в ПЗУ. Программирование при этом приобретает специфический характер, и стандартный Ассемблер помогает только частично. Потому иногда предпочитают манипулировать непосредственно машинными восьмеричными кодами и адресами.

ПВМ с Фокалом снабжены вспомогательным отладочным монитором, позволяющим оперировать с машинными кодами, модификации с Бейсиком могут не иметь такого средства, тогда доступ к ОЗУ, ПЗУ, машинным кодам возможен через функции PEEK и POKE Бейсика, а запуск машинных подпрограмм — через функции пользователя типа USR.

Сведений, приводимых в документации, поставляемой с этими машинами, недостаточно для обучения программированию в машинных кодах или на Ассемблере и тем более для использования ресурсов обрабатывающей системы Бейсика (или Фокала). Приходится разыскивать первоисточники — инструкции к другим машинам, учебники программирования в системе команд DEC (например, для ЭВМ PDP). Многие материалы частями публиковались в журналах (см.список литературы).

Машинное программирование позволяет не только повысить эффективность программ, но освоить дополнительные и специальные режимы работы, обойти недоработки стандартного программного обеспечения. Например, «компенсировать» отсутствие в имеющейся версии Бейсика команды MERGE, позволяющей объединять тексты нескольких программ, записанных на магнитной ленте (или диске центральной машины учебного класса).

Для уверенной работой с машиной надо осваивать несколько разделов сведений:

- структуру машинных команд и, может быть, Ассемблер, в том числе подсистему прерываний, хранение в памяти машины численной и символьной информации;
- операции ввода-вывода, работу клавиатуры и экрана, «драйверы» внешних устройств, регистры связи с внешними устройствами;
- распределение оперативной памяти и ПЗУ, назначение основных системных ячеек (переменных);
- основные алгоритмы работы Бейсика, использование им памяти машины и существенных рабочих переменных;
- блоки, подпрограммы Бейсика (и частично мониторно-драйверной системы), которые могут быть полезными в программах пользователя;
- ассортимент специальных и синтетических приемов программирования, организацию особых режимов работы, варианты связи машинных программ с бейсиковскими и др.

Для начала приведем несколько простых приемов выхода из Бейсика в оперативную память или ПЗУ, доступ к машинным подпрограммам, затем разберем схему обработки Бейсика, использование при этом оперативной памяти и системных переменных, представление информации на разных этапах ее обработки в ЭВМ.

После этого, учитывая стандартную документацию и ранее публиковавшиеся сведения, можно будет описывать специальные приемы смешанного программирования на Бейсике и в машинных кодах, в том числе и из области машинной графики.

1. Простые приемы доступа к машинной памяти

Основные операции прямого доступа к памяти — запись отдельных кодов (чисел) и их массивов, просмотр (чтение) чисел и массивов, очистка заданных массивов (полей) или заполнение их какими-либо константами, копирование (перенос) информации из одного места памяти в другое, сохранение ее на магнитной ленте.

Из описания Бейсика известны функции PEEK и POKE (точнее, оператор и функция); в отличие, например, от машин типа IBM PC эти функции работают с 2-байтовыми словами (первый байт — с четным адресом); внутримашинную информацию в машинах этого типа предпочитают записывать и просматривать в восьмеричном, а не шестнадцатеричном виде, так что два байта записываются максимум шестью цифрами, старшая из которых может быть только нулем или единицей. Если все биты двухбайтового слова заполнены, восьмеричный код будет 177777; с указанием типа восьмеричной константы в Бейсике &O177777.

Для записи по указываемым адресам отдельных кодов (чисел) удобно ввести свой ключ вместо какого-либо не актуального, например, инструкцией:

```
KEY 2,"POKE &O
```

Вызывая затем эту заготовку нажатием двух кнопок, дописывают нужную информацию и пускают в «ход», например, для записи в ячейку 212_8 кода 111_8:

```
POKE &O212, & O111
```

(после этого вновь вводимые нижние строки экрана, а затем весь его фон покроются полосами, отменить это можно командой COLOR 1,0).

Для чтения кодов из отдельных ячеек (слов) можно ввести ключ:

```
KEY 1,"? OCT$(PEEK(&O".
```

Дописывая адрес после вызова заготовки, не надо забывать две завершающие скобки.

В нужных случаях можно заменять функцию OCT$ на BIN$ или HEX$ или же второй параметр для POKE записать с префиксом $H либо как десятичное число (целое).

Сложнее записать по заданному адресу один байт символьной информации, не повреждая парного, особенно если адрес нечетный. Известно, что из пары символов, записываемых в одно слово — в его четный и нечетный байты, первый символ занимает младшие биты, второй — старшие, причем в восьмеричном представлении разряд сотен делится между двумя этими байтами (при установке всех битов младшего байта в единицы восьмеричное значение будет 000377, при единичных битах старшего байта — 177400).

Оператор POKE для записи четного байта (символа A) по адресу 1000_8 может выглядеть так:

```
POKE &O1000, PEEK(&O1000) AND &O177400 OR ASC("A")
```

Прочесть и распечатать тот же байт можно так:

```
? CHR$ (PEEK(&O1000) AND &O377)
```

Логические операции здесь используются для выделения нужных разрядов кода.

Дополнительные затруднения с нечетными байтами вызываются тем, что старший разряд такого байта в арифметической операции (например, умножении) играет роль знака, и Бейсик отказывается работать с числами, превышающими 32767. Хитрая, хотя и громоздкая манипуляция для записи нечетного байта может выглядеть так:

```
POKE &O1000, PEEK(&O1000) AND &O377 OR (((ASC("A")-128)*256)XOR
&O100000)
```

(Записывается байт по адресу 1001_8, а слово имеет адрес 1000_8).

Прочесть и распечатать этот байт можно так

```
:? CHR$ ((PEEK (&O1000) XOR &O100000)/256+128)
```

Такие длинные строки в качестве ключей не введешь, но их можно преобразовать в маленькие подпрограммки; константы (коды и адреса) можно заменить на переменные. Читателям предлагаем дополнительно подсократить приводимые выражения.

Для чтения массивов информации предлагается сравнительно простая программка (чтобы не занимать лишнего места, будем писать по нескольку операторов в строке и опускать их номера, как если бы Бейсик для БК был более современным):

```
1000 INPUT I%
1001 A%=0% : FOR K%=1% TO 22% : ?OCT$(I%)" "OCT$(PEEK(I%)) : I%=I%+2%
     : NEXT : INPUT A% : IF A% THEN I%=A%
1009 GOTO 1001
```

Здесь вначале запрашивается адрес начала выводимого массива, затем последовательно выводится содержимое по 22 строки; каждый раз надо нажимать кнопку «ВВОД» (ВК). Если вместо этого ввести новый адрес (очевидно, восьмеричный), следующая порция информации будет выводиться с этого адреса. Завершается просмотр кнопкой «СТОП».

Для записи информации в массив (с контрольным просмотром) несложная программа может быть такой:

```
1010 INPUT I% : GOTO 1050
1020 INPUT K% : IF K%=&O111111 GOTO 1040
```

```
1030 POKE I%,K%
1040 I%=I%+2%
1050 ? OCT$(I%); TAB(8); OCT$ (PEEK(I%)); TAB(16);
     : K%=&O111111 : GOTO 1020.
```

Здесь тоже сначала запрашивается начальный адрес массива; затем поочередно (строками) выводятся последовательные адреса слов и их старое содержимое; если просто нажать кнопку «ВВОД», старое содержимое остается; в противном случае вводится новая информация. Замечание: коды 111111_8 данная программа не вводит. Вводимую восьмеричную информацию следует предварять префиксом, указывающим на ее восьмеричность. Впрочем, можно вводить и десятичные числа.

Простейшая программа позволяет переносить массивы информации:

```
1100 INPUT I%, J%,N% : FOR K%=I% TO J% STEP 2%
     : POKE N%, PEEK(K%) : N%=N%+2% : NEXT : END
```

Этой программой можно немного сдвинуть массив «вверх» (в сторону уменьшения адресов), «вниз» же можно сдвигать, только если два поля — исходного и результирующего массива — не пересекаются. В противном случае, чтобы не делать более сложную программу, «сдвигают» массив за два приема, используя какое-то временное поле в свободном месте памяти. Зато этим способом можно, предварительно задав значение первого элемента массива, размножить его в нужном количестве, задавая результирующий адрес на 2 больше исходного. После пуска программы надо ввести начальный и конечный адреса исходного массива и начальный адрес результирующего.

При регулярной работе с машинными кодами можно, конечно, использовать более сложные программы, языки, среды и пр.

Читать этими способами можно, конечно, как ОЗУ, так и ПЗУ; записывать только в ОЗУ, но в том числе в системные переменные (защиты памяти нет), в область видеопамяти и др.

На ленте (или диске центральной машины) такая информация сохраняется оператором BSAVE с указанием непосредственных машинных адресов; читается соответственно оператором BLOAD. К сожалению, в отличие от ПВМ типа IBM PC и других в программном режиме эти операторы не срабатывают, так что при необходимости приходится использовать машинные подпрограммы.

Небольшие машинные программы (подпрограммы) и информацию ограниченного объема можно размещать с адреса 1000 до 1400 — 1600 (в надежде, что «стек» не достанет) или с 37000 по 37777 (если не использовать некоторых операций ввода-вывода)*. В других случаях для такой информации отводится область оператором CLEAR n, adr. Здесь adr, как известно, предельный адрес, занимаемый Бейсиком; обычно это начало видеопамяти (40000_8), задавая его меньшим, получаем свободную для машинных программ и информации область, начиная от adr и до начала видеопамяти**. Этот же оператор совместно с переключением режима работы экрана (РП) позволяет либо расширить пространство для Бейсика, либо получить дополнительное место для машинных программ. Изменяя информацию в нескольких системных ячейках, можно «усечь» объем видеопамяти и рабочее поле экрана на четверть или, допустим, треть, получив дополнительное место для других целей (см.Л).

Заставить работать машинные программы (точнее, они будут подпрограммами) можно с помощью функций USRi, где i — цифра от 0 до 9; предварительно оператором вида DEF USRi=&Okkkk задается начальный адрес (точка входа) подпрограммы (kkkk), выход на нее обеспечивается хотя бы фиктивным использованием функции USR : a%=USRi(в%); здесь a% может быть фиктивной переменной, в% — фиктивной переменной или константой (целый тип предполагается в целях экономии). В нужных же случаях в% — входной параметр, a% — рабочее значение функции; такая функция может быть включена, конечно, в арифметическое выражение. В инструкции по Бейсику сказано, как, составляя машинную подпрограмму, разыскать исходный параметр и как устроить передачу Бейсику значения функции (в том числе для различных типов переменных). Известны, однако, искусственные приемы, позволяющие передавать машинной подпрограмме несколько параметров.

Если параметры — целые числа и пользователь может определить их конкретные адреса в машинной подпрограмме, со стороны Бейсика для передачи значений можно использовать функции PEEK и POKE.

Для первого знакомства посоветуем записать в ячейки 1000 и 1002 коды 61515_8 и 207_8 и затем убедиться, что функция USR складывает значение аргумента (целого) с собой, т.е. умножает его на два.

От процессоров машин «Электроника-60» [Л], ДВК и ряда других микропроцессор БК-0010 (К1801ВМ1) отличается в основном отсутствием команд умножения и деления целых чисел.

Процессор — 16-разрядный (в сравнении с 8-разрядными «Микрошами», «Корветами» и даже «Ямахами»), многие команды (но не сложения и вычитания) работают в двух модификациях — как с байтами, так и с двухбайтовыми словами. В процессоре — восемь 16-разрядных регистров (нумеруемых

**От таблицы адресов строк до конца выделенной ОП на область ввода-вывода бронируется 1022_8 байта (адреса в ячейках 2026, 2030, «по умолчанию это 36756_8, 40000_8). Эту область можно использовать для машинных подпрограмм и данных, если не используются операции с магнитофоном (операторы BLOAD, BSAVE не в счет). Однако содержимое ячейки с адресом MAXAD-422_8 (по умолчанию 37356_8) должно быть нулевым, иначе при запуске программы (оператором RUN) система может зависать в ожидании закрытия файла, читаемого с ленты.*

***В некоторых случаях от adr-1000_8 до видеопамяти*

от 0 до 7), которые действуют в основном аналогично, но 7-й регистр (PC) также выполняет роль счетчика адресов команд. Кроме того, 6-й регистр используют с характерной целью — счетчика в стеке памяти (SP). Помимо этого, в процессоре есть 4 одноразрядных регистра*, заполняемых при работе различных команд в зависимости от результатов их операций и играющих роль «признаков» (знака, нуля, переполнения, переноса), которые могут быть использованы последующими командами, например условных переходов. Сами команды (все) можно считать двухбайтовыми, хотя разряды кода операции располагаются в них на разных местах. В зависимости от типа адресации операндов, однако команду могут дополнять одно-два слова, содержащих операнд непосредственно, адрес его расположения,"адрес адреса", приращение адреса и др. Можно заметить, что структура команд разрабатывалась тогда, когда ЭВМ и тем более ПВМ, микроЭВМ имели очень ограниченные объемы памяти; принимались меры для возможного сокращения длины машинных программ. Потому многие команды, при задании соответствующих типов адресации, дополнительно наращивают или уменьшают адрес в задействованном регистре, чтобы проще можно было выполнить следующие команды. Но, с другой стороны, «упаковывая» в два байта команды условных переходов (и «короткого» безусловного перехода BR), обеспечили смещение точки перехода (относительно текущего значения СЧАК) только до 127-128 слов «вверх» или «вниз», а это нередко требует добавления команд «длинных» переходов (JMP), усложнения структуры программы. «Длинная» адресация ограничена 16-ю разрядами, т.е. 64 килобайтами памяти. «Раздвоения» адресов, например на «базу» и «смещение», как во многих других структурах команд, здесь в самих командах не предусмотрено, что усложняет введение постраничной памяти при ее расширении. Слабы команды сдвигов содержимого операндов — команда за один раз дает сдвиг только на один двоичный разряд вправо или влево, и для сдвига на несколько разрядов приходится устраивать циклы. Сложение и вычитание возможны только для двухбайтовых слов. Практически нет команд ввода-вывода. Вместо этого по отдельным адресам установлены не просто ячейки памяти, а регистры, связанные с внешними устройствами. Запись в них (или в их отдельные биты) информации и чтение информации производятся вроде бы на общих основаниях. Фактически под одним адресом могут существовать отдельные — вводной и выводной — регистры, как это, например, сделано для регистра параллельного порта (адрес 177714).

Основные регистры связи описаны в Руководстве системного программиста.

Многое решается программным путем, в том числе программами-драйверами. Так, слова «состояния дисплея» на самом деле нет, в его качестве выступают рабочие (системные) байты с 40_8 по 56_8. Указанная (регистром 177664) область памяти, можно сказать, независимо от процессора постоянно сканируется контроллером экрана и передается на экран, а вывод на экран символов, их подчеркивание, изменение цвета линий и др. — все выполняется программно — пересылкой соответствующей информации в область ОП, отведенную под «видеопамять».

В схеме предусматривается также «ловля прерываний». Один из разрядов в регистре PSW (слово состояния процессора) разрешает или запрещает прерывания работающей программы от внешних устройств или особых ситуаций; свои биты разрешения или запрещения прерываний могут быть в служебных регистрах внешних устройств; для прерываний от различных причин (например, нажатие кнопки на клавиатуре или встреча с неправильной командой) в схему «запаяны» адреса ячеек — «векторов прерываний». Эти ячейки, таким образом, фиксированы, хотя информацию в них можно менять. Первое двухбайтовое слово «вектора» — адрес начала подпрограммы обработки прерывания данного типа, второе слово — загружаемое в регистр PSW, «состояние» на период обработки прерывания. На время обработки прерывания текущее (до «прерывания») значение СЧАК, т.е. содержимое регистра 7, и текущее PSW сохраняются в стеке с соответствующим изменением счетчика стека в регистре 6.

Стек вообще достаточно эффективное и универсальное средство временного сохранения различной информации в отведенной области ОП, связи между различными программами и подпрограммами. Стеки могут реализовываться на разных уровнях и языках, чисто программными приемами. Этот прием заключается в выделении области памяти с некоторым начальным адресом, который может при загрузке данных наращиваться в ту или иную сторону, а при выборке — изменяться в противоположном направлении; таким образом, в ОП или в выделенном регистре процессора заводится счетчик адреса «вершины» стека. Для машинного программирования ПВМ данного типа характерно использование для указателя вершины стека регистра 6 (SP) и, во-вторых, наращивание стека «вверх», т.е. в сторону уменьшения адресов. В текущий момент счетчик указывает на следующий свободный элемент стека. Модификации команд с инкрементом или декрементом адреса в регистре облегчают запись-извлечение из стека. Модификации же команд с заданием смещения адреса позволяют в нужных случаях читать хранящуюся в стеке информацию, не изменяя счетчик. Преимущественное использование регистра 6 связано со схемой процессора; команды обращения к подпрограмме и возврата из подпрограммы, вышеупомянутые прерывания используют стек, адресуемый именно этим регистром, для хранения адресов возврата и других данных. Загружая в регистр 6, однако, тот или иной адрес, можно переходить «со стека на стек». Так, мониторно-драйверная система использует стек, начиная с адреса 776, а Бейсик — свой стек, с адреса 1776.

Использование стеков пользователем должно быть особо осторожным; при нарушении «синхронности» (записали больше, а прочитали меньше, или наоборот) вместо адресов возврата будет читаться ложная информация, появятся другие нарушения — и с программой пользователя и всей программной системой будут происходить всякие непредвиденные вещи. Чаще всего зацикливание или переинициализация системы со стиранием программ пользователя.

Как хранятся числа и тексты в памяти машины? Процессор обрабатывает только целые числа байтами и двухбайтовыми словами. Отрицательные числа хранятся в дополнительном коде, притом старший разряд — единица. Число -1 в восьмеричном виде выглядит как 177777, число -32768 — как 100000. Однако адреса команд и другие специальные данные можно рассматривать как целые числа от 0 до 65535. Машинная команда сложения, впрочем, правильно подсчитывает результат, если операнды интерпретировать как целые без знака, например 32000+32000=64000. Но, с другой стороны, этот же

Точнее — разряды регистра PSW (ССП).

код при учете знака равен 1536_{10}. Машинная команда в этом случае устанавливает признак V результата в единицу, а уж система Бейсика фиксирует ошибку 6.

О хранении символов в последовательных байтах памяти мы уже говорили; некоторые недоразумения порой возникают, поскольку текст мы привыкли читать слева направо, в том же порядке записывать значения последовательных слов; в каждом слове, однако, младший байт соответствует младшим, правым, разрядам, а следующий — старшим, левым, что производит впечатление записи сзаду наперед.

Что касается нормализованных чисел (с плавающей точкой), то их запись и формат «отданы на усмотрение» Бейсика (или Фокала, где, между прочим, такие числа записываются не как в Бейсике).

Может быть, стоит забежать вперед и здесь уточнить представление нормализованных чисел.

Нормализованные числа одинарной и двойной точности занимают соответственно 4 и 8 байтов, в обоих случаях в двух первых байтах располагаются знак числа, порядок и старшие разряды мантиссы, в последующих байтах размещаются младшие разряды мантиссы. Разумеется, числа «двойной точности», а вернее, «удвоенной длины» — не в два раза точнее одинарных, а дают более чем вдвое большее количество точных разрядов чисел при счете.

В первом двухбайтовом слове числа старший разряд — знак числа; при этом отрицательные числа не представляются в дополнительном коде: мантисса всегда записывается как положительная. Разряды 7-14 представляют двоичное значение порядка*, увеличенное на 128, т.е. для чисел, равных или больших (по абс.вел.) 0.5, значение 14-го разряда — единица. Если мысленно инвертировать 14-й разряд, положительные и отрицательные значения порядка будут представлены как обычные целые числа, причем отрицательные числа — в дополнительном коде. Разряды с 0 по 6 — это старшие разряды мантиссы, причем самый старший ее разряд, поскольку для нормализованного числа он всегда единичен, в коде числа не записывается (подразумевается как бы между 6-м и 7-м разрядами). Этот прием позволяет повысить точность счета как раз в два раза. Значение мантиссы — от 0.5 до без малого 1.

Для большей ясности приведем двоичные изображения нескольких чисел.

Десятичное число	Восьмеричная запись старшего слова	Двоичная запись															
		Разряды															
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
		зн. чис.	порядок								мантисса						
0.5	40000	0	1	0	0	0	0	0	0	0	(1)	0	0	0	0	0	0
1	402000	0	1	0	0	0	0	0	0	1	(1)	0	0	0	0	0	0
5	40040	0	1	0	0	0	0	0	1	1	(1)	0	1	0	0	0	0
—10	141040	1	1	0	0	0	0	1	0	0	(1)	0	1	0	0	0	0
0.1	37314	0	0	1	1	1	1	1	0	1	(1)	1	0	0	1	1	0

Ноль изображается всеми нулевыми разрядами.

3. Мониторно-драйверная система

Вместо операционной системы для этих маленьких машин в одной из микросхем ПЗУ (с адресами 100000-117777) «зашита» мониторно-драйверная система, включающая пусковой монитор с ограниченными возможностями, программы-драйверы, обслуживающие периферийные устройства, интерпретацию ЕМТ-команд (обращение к драйверам обычно и происходит через команды ЕМТ). Системные команды ЕМТ, от ЕМТ 4 до ЕМТ 50, описаны в руководстве системного программиста. Сами по себе блоки их обработки (драйверы) могут быть достаточно сложными, включать внутренние подпрограммы, таблицы и другие данные. При работе монитора и драйверов используются рабочие ячейки с младшими адресами (от 4 до 277_8), назначение которых описано в [Л], и стек от слова 776 (в сторону уменьшения адресов). Для вывода на экран изображений символов с адреса 112050_8 по адрес 116075_8, например, расположена таблица их растровой кодировки; для изображения на экране отрезков прямых в драйвере для ЕМТ 32 имеется специализированный алгоритм интерполяции, превращающий заданный отрезок в набор точек, засылаемых в нужные биты видеопамяти.

Начало этой области памяти устроено так.

Адреса	Содержание
100000 — 100003	Переход к адресу 100260 (начало монитора)
100004 — 100051	Таблица адресов драйверов — программ обработки прерываний от ЕМТ 4 до ЕМТ 50
100052 — 100111	Продолжение таблицы для старших кодов ЕМТ, однако «зашиты» адреса в ПЗУ, которые имеющимися ПЗУ Бейсика использованы быть не могут
100112 — 100137	Блок расшифровки команды ЕМТ и перехода на обрабатывающую программу
100140 — 100257	Обработка ЕМТ 14 (инициализация драйверов)
110260 — 100737	Пусковой монитор: в начале инициализируется стек и драйверы, затем в виде подпрограммы вызывается система обработки Бейсика (с адреса 120000, это также может быть интерпретатор Фокала или другая система в ПЗУ) Если ПЗУ с адреса 120000 не установлена или специальной командой осуществлен выход на монитор, обработка продолжается с адреса 100274

* *Номера разрядов считаем справа налево, начиная с нуля.*

100740 — 101007 Обработка ЕМТ 4 : инициализация драйвера клавиатуры
101010 — 101724 Обработка ЕМТ 6 : чтение символа с клавиатуры
. . . . . .

Расположение остальных драйверов можно посмотреть в таблице адресов.

4. Общая схема работы системы Бейсика

Система обработки Бейсика расположена в трех схемах ПЗУ и занимает адреса с 120000 по 177531 (дальше, до адреса 177777, поле служебных регистров) и тем исчерпывается доступная адресация в 64 килобайта.

Бейсик использует по мере надобности системные «драйверы» и монитор (обычно через команды ЕМТ), системные ячейки с начала ОП по 371; стек переключается на адреса от 2000_8 «вверх» (максимально до адреса 1000); с адреса 2000 по 3051 — рабочие ячейки Бейсика, частично упомянутые в [Л], а подробнее рассматриваемые ниже; с адреса 3052 и до начала видеопамяти (адрес 40000) или до предельного адреса, «разрешенного» командой CLEAR, располагается текст программы на Бейсике, далее «протранслированный» объектный код, другие таблицы и данные. Чтобы точнее описать распределение памяти, надо ознакомиться с общей схемой работы системы Бейсика.

Известно, что языки типа Бейсик обычно обрабатываются в режиме интерпретации, что, в частности, позволяет быстро запускать подготовленные программы и сочетать программный (или косвенный) режим работы с непосредственным. Однако более сложные программы, с интенсивным использованием подпрограмм и циклов, обрабатываются в режиме интерпретации медленно, потому в последнее время ряд фирм переходит на режимы компиляции программ в машинные или близкие к ним программы или на какие-то промежуточные схемы обработки. Так, для машин типа IBM PC распространены «Turbo-Basic» и «Quick-Basic», в чем-то уже приближающиеся к Фортрану, Паскалю и им подобным языкам. Для машины с малым объемом памяти принятая сложная схема обработки оказалась неудачной ввиду ее неэкономности в использовании оперативной памяти, хотя удается повысить скорость работы программ. Правда, после пуска объемистой программы определенное время уходит на ее компиляцию (если одновременно в памяти записано несколько программ, допустим, большая основная и короткая дополнительная, то запустить можно короткую программу, но компилироваться будут, увы, все находящиеся в памяти программы). Непосредственно вводимые операторы обрабатываются отдельно, при этом сохраняется доступ к сформированным перед этим программой данным.

Обработка программы на языке Бейсик — двухступенчатая. Сначала программы с подпрограммами, записанные в памяти, компилируются в некоторый промежуточный ("объектный", иногда, наверное, по недоразумению называемый «шитым») код, затем этот код — без каких-либо дополнительных воздействий пользователя — начинает интерпретироваться специальной мониторной частью обрабатывающей системы. Компилирующая часть не совсем обычная в том смысле, что не создает машинных программ; промежуточный код состоит в основном из адресов обрабатывающих подпрограмм мониторной части, адресов данных, самих непосредственных данных (константы в исходном тексте). В промежуточном коде переменные уже не ищутся по их именам, хотя в особых случаях имена подвергаются анализу (например, для определения типа переменной). Изредка, увы, из объектного кода производятся обращения к исходному тексту (например, для чтения текстовых констант); использование объектного кода без исходного текста и размещенных по абсолютным адресам переменных не предусматривается. Объектный код также нельзя перемещать в памяти потому, что он содержит абсолютные адреса точек переходов. Пересылка параметров, подсчет выражений и функций на этапе интерпретации выполняются с широким использованием стека. Имитируется как бы нуль-одноадресная виртуальная машина. Загружается в стек один параметр, другой параметр (или константа), затем вызовом специальной подпрограммы выполняется их сложение или умножение, «верхний» операнд из стека стирается, вместо «нижнего» образуется результат, который дальше может быть использован в следующей операции или сохранен в памяти... Нужный порядок действий определяется на этапе компиляции и превращается в нужную последовательность операций со стеком и операндами в объектном коде. Вместо массивов, текстовых констант и переменных в стек загружаются, конечно, их адреса. В некоторых случаях используются дополнительные стеки.

Как образуется исходный текст программы на Бейсике?

При наборе текста оператора (строки) работает подпрограмма «редактор» IVEIL* (нач.адрес 121052); информация при этом накапливается в буферной области BUF (адреса 2422-3021). После нажатия кнопки «ВВОД» и выхода из редактора информация анализируется; если это непосредственный оператор, он отрабатывается одним способом, а если строка программы (начинающаяся с номера) — она заносится в следующее свободное место области, отведенной для исходного текста программ (первая строка заносится с адреса 3052); при этом номер строки не заносится, а оператор (команда) кодируется специальным однобайтовым номером; символ «ВК» в конце строки (код 12_8) заносится. Номер строки и начальный адрес ее расположения заносятся в таблицу TABNUM, которая располагается в конце отведенной области памяти (точнее см.ниже) и наращивается вверх. Для каждой строки отводятся три двухбайтовых слова: адрес начала оператора объектного кода (до пуска нулевой программы), адрес начала исходного текста строки и номер строки. Эта таблица формируется упорядоченной по номерам строк; строки с большими номерами располагаются выше. При стирании строки таблица «поджимается», при добавлении новой, с промежуточным номером, нужная часть таблицы с большими номерами приподнимается. Изменения исходного текста производятся иначе: новые строки в порядке их введения и независимо от номеров записываются в хвост массива. Если строка редактируется или заново пишется строка с

---

* *Имена подпрограмм и блоков условны.*

уже существовавшим номером, запись происходит аналогично, но из середины массива старая строка изымается, массив поджимается вверх.

При выводе строк на экран (командой LIST или для редактирования) коды операторов дешифрируются и заменяются словесными. Вот почему, если оператор был введен строчными буквами или в сокращенном виде, без пробелов, восстанавливается он в стандартном виде.

Пользователи этих машинок знают, что большинство операторов можно записывать двумя-тремя буквами как в начале строки, так и иногда в ее середине (для сложных операторов); во многих ситуациях оператор можно не отделять пробелом от последующей информации, например от имени переменной. Но, с другой стороны, это порой приводит к недоразумениям, ошибкам, как, впрочем, использование имен переменных, содержащих более чем 2 символа (не считая специальных, указывающих тип переменной). В качестве имен нельзя использовать не только ключевые слова как таковые, но и сочетания их первых двух-трех букв. Нельзя, например, ввести переменную IN, поскольку это начало оператора INPUT. Впрочем, путаница возникает в основном, когда имя переменной стоит слева от знака равенства в операторе присвоения; спастись можно, явно записав тип этого оператора — LET (что сейчас, как правило, никто не делает). При искажении написания оператора Бейсик ошибается. Так, оператор INPAT I превратится в INPUT AT I (появится синтаксическая ошибка).

Оператор IFI=A THI=4 — правильный (переменной I может быть присвоено значение 4), оператор IFI=A THIN=4 — неправильный. Оператор IFA THLETIN=4 — опять правильный. При записи строк в массив исходного текста числовыми кодами заменяются только первые (ведущие) операторы.

При записи текста программ можно сокращенно писать операторы (ключевые слова), но запись ведущих (первых в строке) операторов на длину программы, как видим, не влияет.

Таким образом, при записи и редактировании программы-источника (на яз. Бейсик) свободная область памяти заполняется «с двух сторон», навстречу друг другу. Эта тенденция прослеживается дальше.

При пуске программы (командой RUN) на этапе компиляции составляется «объектный код», расположенный после текста-источника (начиная со свободного четного байта) «сверху вниз», и заводятся переменные, описание и содержимое которых (сначала нулевое) размещается «перед» упоминавшейся таблицей адресов; эти данные наращиваются снизу вверх. Подробнее к представлению данных вернемся, пока заметим, что для массивов и символьных переменных заводятся описатели — дескрипторы с именами, но еще не заполненными адресами, где будут расположены сами данные.

Заполняются адреса в таблице TAB (для объектного кода).

После компиляции система автоматически переходит к отработке объектного кода (интерпретации), хотя при экспериментах ее можно «обмануть»: если первый выполнимый оператор END или STOP, обработка объектного кода тут же заканчивается (хотя весь код сформирован).

Перед интерпретацией вслед за «объектным кодом» система отводит в памяти область для символьных переменных — по умолчанию 256 байтов, но оператором (командой) CLEAR эту область можно изменить; далее отводится область для самих массивов; символьные переменные и массивы накапливаются динамически, например, массивы по мере отработки операторов DIM или явочного использования их элементов. Такая инициализация массива или текстовой переменной приводит к заполнению в дескрипторе адреса его расположения.

«Снизу» используемая память тоже динамически наращивается. «Перед» областью задания переменных располагаются подобия стеков для циклов и для возвратов из подпрограмм. Притом если несколько циклов расположены последовательно, область, занимаемая для них, не растет; она увеличивается при появлении вложенных циклов.

Ошибка 7 — переполнение памяти — фиксируется, когда заполнение памяти «сверху» и «снизу» смыкается.

В виде таблицы использование памяти Бейсиком можно изобразить так.

Адреса или условные имена областей	Содержание
1000-1777	Стек
2000-2421	Системные переменные
2422-3021 (BUF)	Буфер ввода (с клавиатуры)
3022-3051 (BUFOUT)	Буфер вывода
3052, до TXEND(TEXT)	Текст — источник программы
От TXEND до STREG	Объектный код программы (начиная с четного адреса)
От STREG, на заданную величину	Область для символьных переменных
От конца области символьных переменных до ENDKOD	Область для массивов
	Свободная область (если остается)
От CICL до LIMIT	Дополнительный стек для циклов
От LIMIT до LENT	Дополнительный стек для возвратов из подпрограмм
От LENT до TABTOP	Область переменных (и дескрипторов)
От TABTOP до FCB	Таблица TAB номеров строк и адресов программы
От FCB до HIMEM	Область для обмена данными с магнитофоном; занимает 1022_8 байта (фактически буфера ввода-вывода для команд LOAD, SAVE и др.)
HIMEM	Начало видеопамяти (по умолчанию 40000) или предельный адрес, установленный командой CLEAR

Конкретные адреса устанавливаются системными переменными. Разговор о них уже начинался в [Л]. Ниже приводим более полные сведения (имена переменных условны).

Адрес	Усл.имя	Назначение
2000	LIG	Индекс режима работы системы
2002	TXEND	Указатель (адрес) конца текста — источника программы
2004	STREG	Указатель конца объектного кода, начало области символьных переменных
2006	STSIZ	Длина области символьных переменных
2010	STFRE	Начало свободной части в области символьных переменных
2012	STLEN	Длина свободной части области символьных переменных
2014	ENDKOD	Конец области массивов, а если их нет — области символьных переменных
2016	CICL	«Вершина» стека для циклов
2020	LIMIT	«Вершина» стека для возвратов из подпрограмм
2022	LENT	Начало области переменных
2024	TABTOP	«Вершина» таблицы номеров и адресов строк (TAB)
2026	FCB	Начало области для связи с магнитофоном
2030	HIMEM	Указатель предельно используемого адреса ОП
2032-2044		Различные рабочие переменные
2046	HALT	Адрес (в объектном коде) останова программы для оператора CONT
2050		«Флаг» режима трассировки
2052		«Флаг» режима автонумерации
2054		Приращение номеров строк для автонумерации
2056	CURLIN	Указатель текущего положения курсора
2060-2077		Рабочие ячейки
2100-2123	TABUS	Таблица начальных адресов для машинных подпрограмм USRi
2124-2137		Рабочие ячейки
В т.ч. 2126	PF	Адрес блока параметров для функции AT
2134	SAVJMP	Хранение адреса возврата из подпрограммы
2140-2411		Тексты для ключей
2414		Адрес драйвера записи
2416	I0DEV	Указатель устройства ввода-вывода
2420	1NK	Буферная ячейка для оп. INKEY$

5. Имена переменных и дескрипторы

На имя переменной или массива отводится два байта, притом делается попытка уместить не только два символа имени, но и тип переменной (который может определяться третьим или вторым — служебным символом, добавляемым к имени), а также указание на то, что именем определяется массив или функция. «Выгадывание» производится за счет отбрасывания старших разрядов байта — 3 разрядов для первого символа имени и 2 разрядов для второго. Это оказывается возможным потому, что первый символ может быть только латинской буквой (причем заглавные и строчные буквы не различаются), а второй — латинской буквой или цифрой.

Разряды двухбайтового слова разделяются так (вот здесь можно сказать: «шитое» — из кусков — имя):

бит	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Указатели типа			Первый символ имени					Указатель массива	Указатель имени функции	Второй символ имени					

Если второго (не служебного) символа в имени нет, соответствующие разряды нулевые. Для простых переменных разряды 6,7 — нулевые. Что касается битов 15-13, для разных типов переменных соответствие следующее:

двойная точность (служебный символ # или его нет) — 000
одинарная точность (служебный символ !) — 001
целые переменные (служебный символ %) — 010
текстовые переменные (служебный символ $) — 111

Так, имя AB% превращается в код 40402_8.

Дескриптор массива занимает 6 байтов. Первое двухбайтовое слово — его имя (разряд 7 — единичный), следующее слово — адрес его местоположения (заполняется динамически), следующее слово в младшем байте содержит размерность массива (количество индексов), в старшем байте — длину одного элемента массива (2 — для целочисленных, 8 — для массивов с «длинными» нормализованными числами, 6 — для массивов текстовых переменных). Адрес местоположения массива (в соответствующей области) указывает на заголовок массива; заголовок содержит столько двухбайтовых слов, сколько ин-

дексов имеет массив; в каждом слове — максимальное значение индекса, увеличенное на 1, поскольку минимальное значение индекса — нулевое, причем начиная с правого индекса. Заметим, что в системе Бейсика — ошибка, правильно работают только одно- и двухиндексные массивы. После заголовка в нужном порядке, по оговоренному количеству байтов, записываются элементы массива. Однако для массива символьных переменных элементами массива являются «дескрипторы второго порядка». Каждый содержит 3 байта. Первый из них — длина данного символьного элемента (в байтах), два следующие — независимо от того, у которого четный или нечетный адрес, — указывают начало расположения символьного элемента (подмассива) в области текстовых переменных.

Дескриптор текстовой переменной, кроме имени, содержит два двухбайтовых слова — длину переменной (в байтах) и адрес ее расположения в области текстовых переменных.

Описание функции пользователя (типа FN) содержит имя (6-й разряд единичный) и два двухбайтовых слова. В первом — адрес дескриптора в определенном месте объектного кода, где задается эта функция; дескриптор занимает несколько слов; первое — количество определенных формальных параметров, в следующих словах (по количеству параметров) специальными кодами заданы типы этих параметров.

Во втором слове описания функции динамически (при вызове функции) формируется адрес возврата в объектный код, где вызывалась функция. При вызове функции из объектного кода после адреса точки входа в подпрограмму FNCALL (156046_8) также размещается дескриптор, содержащий аналогичные данные об уже фактических параметрах (если будет замечено расхождение — выведется сообщение об ошибке). Сами же параметры для функции передаются через стек.

6. Особенности работы объектного кода

Объектный код в данном случае представляет собой псевдопрограмму, командами в которой служат адреса обрабатывающих подпрограмм; в ту же последовательность включаются адреса данных и сами данные непосредственно. Константы из текста-источника, как правило, перекопируются во вторичный код (с переводом во внутримашинное представление), переменные в объектном коде определяются адресами, причем адреса указывают не на имена, а на содержимое переменной или на начало дескриптора. Текстовые константы остаются в тексте-источнике, из объектного кода на них производятся ссылки по адресам. В общих чертах обработка объектного кода описана в [Л]. Там же приведены частичные таблицы адресов обрабатывающих подпрограмм. Уточним некоторые детали.

Более чем 2-байтовые числа располагаются в объектном коде «вверх ногами», т.е. сначала записываются слова с младшими разрядами мантиссы, а затем старшими разрядами и порядком. Текстовые переменные пересылаются в стек и из стека так же, как 4-байтовые числа подпрограммами с адресами 156214_8 и 156330_8, поскольку пересылаются фактически их 4-байтовые дескрипторы. Пересылками самих текстовых переменных из области их хранения в область хранения занимаются такие подпрограммы, как SQUEE, NEWSTR (160634, 156672) и др.

Оператор DIM обрабатывается подпрограммой ARRAY (160226), которой дается ссылка на дескриптор массива и количество индексов. Предварительно в стек засылается максимальное значение индекса, заданное в операторе DIM. Аналогично выбирается значение элемента массива: в стек засылается значение индекса (или последовательно значения двух индексов), затем подпрограмме INDEX (160056) дается ссылка на дескриптор и количество индексов; эта подпрограмма формирует в стеке значение элемента.

Особым образом формируется обработка циклов, условных переходов, вызов функций и тому подобные действия. Полное описание было бы слишком объемистым для журнала.

Сам принцип составления подобных объектному коду «псевдопрограмм» изобрели, конечно, не разработчики данной версии Бейсика; иногда такая форма называется польской. Подпрограммы, обрабатывающие такую запись, завершаются командами перехода — обычно по регистру 4: JMP @(R4)+(134_8); благодаря автоинкрементации адрес в регистре 4 увеличивается и устанавливается на следующий элемент псевдопрограммы, так что переход СЧАК осуществляется прямо к началу следующей обрабатывающей подпрограммы, адрес которой выбран из элемента псевдопрограммы. Из обрабатывающей подпрограммы можно читать данные (адреса или числа), выбирая их из псевдопрограммы (например, командой MOV (R4)+,R1 (12401_8), которая тоже наращивает счетчик в R4). Для «входа в режим» обработки псевдопрограммы в R4 заносится адрес ее начала, затем выполняется команда 134.

Из машинной программы ("настоящей") можно перейти к продолжающему ее псевдокоду экономнее, обратившись к подпрограмме STAND, состоящей всего из двух команд:

```
166562: 5726     (TST (SP)+)
166564: 134      (JMP @(R4)+)
```

Вызов этой подпрограммы производится двумя словами: 4437, 166562. Чтобы выполнить обратный переход к машинным кодам, в очередном слове псевдопрограммы записывают адрес следующего слова ОП. Компактность такого приема позволяет им достаточно часто пользоваться (он неоднократно используется и в обрабатывающей системе Бейсика, например при вычислении функций).

7. Итоги и возможности

Итак, пользователю, начинающему осваивать машинную кодировку и внутренние ресурсы Бейсика, предлагается «смесь» из настоящих машинных команд и псевдокода рассматриваемого типа. При этом возможны разные акценты и сочетания:

добавить к машинной программе псевдопрограмму для использования готовых подпрограмм «плавающей арифметики», вычисления функций и других из ПЗУ системы Бейсика;

- подготовить наборы своих различных программ, расширяющих возможности машинной кодировки (т.е. возможности процессора), ими пользоваться через псевдокод;
- к этой системе добавить свою TRAP-обработку или перехватывание, с собственной расшифровкой псевдокоманд (с несуществующими кодами), как это используется, между прочим, в интерпретаторе Фокала;
- использовать фрагменты объектного кода, полученного трансляцией (бейсиковских программ), с той или иной их доработкой (например, изменением адресов переменных и адресов точек перехода); как упоминалось, при некоторых ограничениях такой объектный код может работать без программы-источника.

Дополнением некоторых служебных программ можно для типовых задач сокращать объем псевдокода; вместо нуль-одноадресной имитируемой (виртуальной) машины перейти к одно-двухадресной.

В данной версии Бейсика единственная возможность использовать формальные параметры, записываемые списком, в скобках, имеется в функциях пользователя FN; однако у этих функций другие жесткие ограничения: функция должна быть описана фактически одним арифметическим выражением и дает один результат; нельзя этим способом, например, порождать графические образы. Специальная методика с использованием машинных подпрограмм (типа USR) позволяет совместить эти потребности.

Наборы машинных подпрограмм позволяют расширить возможности Бейсика, например, в области машинной графики; моделировать, например, «резиновую нить», не стирающую элементов изображения, по которым она проходит, получить быструю перекопировку или смещение фрагментов изображения (наподобие операторов PUT и GET в Бейсиках для ПВМ типа IBM PC) и так далее.

В целях вмещения в память машины более объемистых программ возможны синтетические приемы с делением их на части и поочередной отработкой, в том числе с использованием внешнего носителя.

В составе учебных классов, содержащих по нескольку ПВМ этого типа, возможно для относительно более сложных задач их попарное использование с соединением через параллельные порты; притом различным образом могут распределяться их память и назначаться функции экранов (один из них может работать в режиме РП); в итоге общий алгоритм может быть чуть ли не втрое сложнее.

Машинное программирование актуально и при подключении к ПВМ нестандартных устройств — управляющих рукояток, дополнительных пультов и датчиков, графического печатающего устройства или графопостроителя.

8. Примеры

В качестве несложного примера приведем текст машинной программы, перекопирующей прямоугольный фрагмент изображения с экрана в другое место; машинная подпрограмма работает примерно в 6 раз быстрее, чем двойной цикл на Бейсике с использованием операторов POINT и PSET. В ней используется подпрограмма системы Бейсика COLOR, начинающаяся с адреса 126402 (в регистрах 1 и 2 — координаты X, У точки, в регистре 0 образуется код цвета, однако в специальных значениях, соответствующих символам управления цветом для экрана, что требует корректировки). Команда EMT 30 — обращение к системному драйверу вывода точки, ему информация передается в регистрах 0,1,2. Для возможного упрощения исходные данные, определяющие размещение исходного и результирующего фрагментов, размещаются прямо «в теле» программы, чего не встретишь в программах, «зашитых» в ПЗУ. Для занесения исходных данных по нужным адресам в Бейсике используют операторы POKE, для обращения к подпрограмме — DEFUSR=&O37000 и A%=USR(0%).

Координаты X1, У1 левого верхнего угла прямоугольника исходного фрагмента заносятся по адресам 37002_8 и 37004_8, координаты диагонального угла — по адресам 37060_8 и 37050_8, вектор переноса, т.е. разности координат для всех перекопируемых точек и их отображений, заносится по адресам 37024 и 37030. Текст подпрограммы (с адреса 37000) следующий:

```
37000:  12701      0   12702      0
37010:   4737 126402  162700    224
37020:   5400  62701       0  62702
37030:      0 104030  163701  37024
37040: 163702  37030    5202  22702
37050:      0 100356    5201  22701
37060:      0 100350     207
```

Еще быстрее можно скопировать или запомнить в отведенном месте памяти фрагмент изображения, если пересылать информацию не по точкам (по одному или двум битам), а по байтам. При этом, правда, в горизонтальном направлении начало и конец копируемого моля должны задаваться координатами, кратными 4. Программу можно сделать так, что если значения координат другие, она добавит слева и справа дополнительные копируемые точки, но перемещать этим способом фрагмент изображения в горизонтальном направлении можно только шагами, кратными 4 точкам (мы имеем в виду режим работы «32 символа в строке»).

Ниже приведем подобную программу, выполняющую две функции: сохранение фрагмента изображения в заданной области памяти и восстановление из указанной области в то же или другое место экрана. На примере этой программы продемонстрируем использование некоторых системных ячеек и стека, заодно также — искусственный прием передачи параметров машинной подпрограмме (через стек) взамен использования операторов POKE.

В программе на Бейсике адрес машинной подпрограммы задается, как и прежде, адрес начала области памяти для запоминания фрагмента засылается через функцию POKE в ячейку 37002 (если он

постоянный — может быть включен как константа в кодировку машинной подпрограммы). Вызов подпрограммы выполняется таким фиктивным арифметическим (точнее, логическим) выражением:

A%=X1%=(Y1%=(X2%=(Y2%=USR(I%)))).

Здесь A% — фиктивная переменная, X1% и Y1% — константы, переменные или выражения (обязательно целые), задающие левый верхний угол прямоугольного поля для фрагмента, X2%,Y2% — аналогично — координируют правый нижний угол. I% — индекс, указывающий тип операции; если это 0% — выполняется сохранение изображения, в противном случае — его восстановление. Для восстановления запоминавшегося изображения значения X1% и Y1% могут быть другими (чем при запоминании), но разности X2%-X1% и Y2%-Y1% должны сохранить свою величину (правда, вторая разность может быть меньше, чем при запоминании). Записанное выражение синтаксически корректно, логические операции типа = безопасны в том отношении, что не вызывают переполнения при любых значениях операндов, а вложенные скобки заставляют параметры подряд накапливаться в стеке. После входа в машинную подпрограмму, если указатель стека в регистре 6 имеет значение к, вид стека такой:

Адрес	Содержимое
к	адрес возврата из машинной п/п (в среду Бейсика);
к+2	I% (параметр USR; этот адрес — в рег. 5)
к+4	Y2%
к+6	X2%
к+10	Y1%
к+12	X1%
....	....

Текст подпрограммы с пояснениями:

37000:	1270	авт.				'засылка адреса начала массива в R4
37004:	6266	6	6266	6		'X2%\4 (отн.номер байта конца строки). Операнд находится в стеке
37014:	16603	10				'пересылка Y1% в R3
37020:	16601	12	6201	6201		'пересылка X1% в R1 и вычисл. X1%\4
37030:	10300	12702	6	6300	77202	'Y1% в R0, вычисл. Y1%*64
37042:	60100					'Y1%*64+X1%\4 — номер байта относит. начала рабочего поля видеопамяти;
37044:	63700	204	42700	140000		'в яч. 204 — относ. адрес начала рабочей зоны видеопамяти; 15 и 14-й разряды подавляются вследствие кольцевого строения видеопамяти
37054:	63700	202				'в яч.202 — начальный адрес видеопамяти; в R0 — абсолютный адрес нужного байта из видеопамяти
37060:	5715	1012				'переход к адресу 37110, если было I%≠0%
37064:	112024	5201				'пересылка байта из видеопамяти в отведенный массив и наращивание адресов в R1 и R4
37070:	26601	6	100355			'проверка X2%≥X1% и усл.переход к адресу 37020
37076:	5203	26603	4	100345		'переход к след.строке, проверка по Y2% и усл.переход к ад. 37020
37106:	207					'выход из п/п
37110:	112420	765				'при восстановлении изображения: пересылка байта в видеопамять и переход к адресу 37070

Подпрограмма переместима, только следует учесть переадресацию ячейки 37002 (для программы на Бейсике).

Для небольших фрагментов подпрограмма работает доли секунды, ее можно применять, например, для временного вывода на экран табличек, фигур с последующим восстановлением изображения по цепочке: запомнили старое содержимое экрана — вывели временное содержание (тоже копированием либо выводом букв, цифр, графических элементов) — восстановили «старое» содержание.

Заменяя по адресу 37110 команду пересылки логическим сложением или «исключающим или» (придется использовать две команды), можно получить эффект «наложения» запоминавшегося фрагмента на фоновое изображение.

малое предприятие Агентство "ЗИГЗАГ"

ЗИГ Агентство ЗАГ

г.Москва, ул. Большая Полянка 11/14, тел. 150-87-03

МАЛОЕ ПРЕДПРИЯТИЕ «КОНУС»

и АГЕНТСТВО «ЗИГ - ЗАГ»

предлагают русифицированные справочные руководства мгновенного доступа (формата Norton Guides), обучающие программы для IBM PC, методические разработки, книги по локальным вычислительным сетям NETWARE 286

Справочные руководства мгновенного доступа

Название	Цена, руб.
CLIPPER	960
Clipper tools one	461
The Library for Clipper (Planet Sofrware)	231
SQLBASE for Clipper	231
Операционная система QNX	1150
RBASE-5000	290
EXEL for Windows:	
1. Gettingg Started and Quick Reference	770
2. Macro and Utilities MS-KERMIT	770
Novell SFT Advanced Netware 286 v. 2. 15:	
1. Users Reference Set	385
2. Supervisor's Guide	385
3. Supervisor Reference	385
4. Console Reference	385
5. Novell Installation Set	385
6. Novell Supplement's	385
MS DOS для программистов	385
Macro-Assembler	346
Turbo-Basic	346
Turbo-Pascal	385
FAST WIRE	155
LAPL-LINK	155
Procomm	155
Справочник инженера-электроника по эксплуатации персональных компьютеров	461
Multiplan 4	346
Язык С для ОС QNX	461
Norton Commander	193
Norton Guides	155
Norton Utility	155

Название	Цена руб.
РЕБУС	500
PCTOOLS	346
LEXICON v.6.57	155
BRIFF	155
Multiplan v.3.O	308
BTRIEVE	576
Super Calk v.5.O	576
Xtalk	193
Disk Manager	193
RS-232	116
Dlink-IV	195
OS/2	461
PKARC	193
Fox Base	346
ASSEMBLER	461
LOTUS 1-2-3	576
Turbo-Assembler	423
MS DOS v.4.1	395
Vitamin-C	193
MS C	231
RBASE SYSTEM V.:	
1. Команды	269
2. Ошибки	231
PKARC	155
GW-BASIC	155
QNX	461
REVELATION	346
Quattro Pro	576

Обучающие программы

Название	Цена, руб.	Название	Цена, руб.
DOS Help v.3.2O	461	RBASE	807
Norton Integrator	423	Dbase-IV	461
Super Calk v.5.O	423	Frame-Work-III	461
LOTUS 1-2-3	423	Btrieve	461
GW-MOUSE	385	MS C	385
Операционная система реального времени QNX	1150	DOS Help v.3.3O	120
Работа в Multyplan 4	385	Microsoft Quick C	461
Foxbase Pro Lan	461	Quattro Pro	807

Методические разработки

Название	Цена (руб.)
Язык АССЕМБЛЕР для персональных ЭВМ	384
Язык программирования турбо-Паскаль версия 5.0	192
Программный комплекс создания руководств мгновенного доступа	154
Язык программироапния Турбо-Си	346
Программный комплекс для исследования и корректировки дисковой информации (Norton Utilities Main Program)	154
Монитор сервисных программ NORTON	154
Субд DBASE III PLUS (РЕБУС)	231
Программирование в R: BASE System V	346
Vitamin C	231
Межмашинное взаимодействие персональных ЭВМ по стыку RS-232C	192
Работа на персональных ЭВМ с пакетом обслуживающих программ PC Tools Deluxe 4.21	231
Экспертные системы для персональных ЭВМ	308
Пакет межмашинной коммуникации XTALK	192
D-Link. Сетевая программа 4.1. Руководство пользователя.	192
Сетевая программа PROCOMM	308
Основы R: BASE SYSTEM V	346
Субд FoxBASE+ Версия 2.00	500
Система межмашинной коммуникации LAP-LINK	
Операционная система ОС-2	308
Система Парадокс. Руководство пользователя.	346
Руководство по эксплуатации системы BTRIEVE	346
Язык манипулирования данными Клиппер	346
Clipper Tools One	461
Использование программного комплекса FASTWIRE II для связи персональных ЭВМ	173
Утилиты создания архивов PKWARE	730
Руководство по операционной системе QNX	576
Сетевое программное обеспечение NOVELL Netware Advanced	692
Протокол передачи данных Kermit	538
Работа с пакетом Word 5	500
Работа с пакетом Ventura Publisher	346
Работа с ОС LanSmart	461

Книги

1. Комплект книг:

1) Локальные вычислительные сети Netware 286

2) Технические средства ЛВС Netware 286

3) Установка ЛВС Netware 286

Стоимость комплекта 148 руб.

2. Работа в сети Netware 286 — 55 руб.

Условия поставки

1. Предоплата на р/с 468550 в Мосмежрегионбанке г. Москвы, МФО 211899

2. Стоимость дискет — договорная. Возможно выполнение заказов на дискетах заказчика.

3. В наш адрес выслать письмо с вашим почтовым адресом, перечнем программ и пособий и копию платежного поручения.

Наш адрес: 125171, Москва, Ленинградское ш., д. 18

Телефон для справок: 150-85-07

Система управления базами данных ИРИС (Инструмент Разработчика Информационных Систем) — сравнительно молодой программный продукт, разработанный в нашей стране. Используемая в нем модель организации данных и удобные диалоговые средства поддержки, выполнения и разработки делают СУБД незаменимым помощником для пользователей IBM PC — от начинающих до программистов-профессионалов.

ОЧКОВ В.Ф., ВАСИН В.А.

СУБД ИРИС

С появлением персональных компьютеров четко определились три наиболее общие области их делового применения: подготовка текстов, проведение вычислений в электронных таблицах, ведение различных баз данных (здесь, естественно, не приводятся другие сферы применения персональных компьютеров, имеющие узкоспециальную ориентацию). Как правило, работа с различными текстовыми процессорами и электронными таблицами не встречает серьезных затруднений даже у начинающих пользователей из-за того, что при этом практически не требуется никаких специальных навыков. Иначе обстоит дело с базами данных, работа с которыми предполагает в качестве первого шага создание собственно базы данных, отвечающей потребностям пользователя. В большинстве конкретных приложений пользователь-непрофессионал в программировании просто не в состоянии сделать даже этот первый шаг, не говоря уже о разработке информационно-поисковой системы (ИПС) в целом.

Одним из основных требований к ИПС является способ представления информации пользователю. В идеальном случае вся необходимая информация должна подаваться в виде документа (платежной ведомости, сметы, личного дела, библиографической карточки), с которым пользователь ИПС (бухгалтер, плановик, библиотекарь) привык иметь дело в своей повседневной деятельности. Такой подход позволяет достичь наиболее хороших результатов при работе с ИПС, безболезненно преодолеть чисто психологический барьер при работе с компьютером. Эта, казалось бы, простая задача таит в себе массу сложностей, преодоление которых требует определенной подготовки в области программирования.

Дело в том, что все известные и широко распространенные системы управления базами данных (СУБД) используют так называемую реляционную (табличную) форму организации данных. Это означает, что данные можно уложить в одномерную таблицу, каждая строка которой содержит данные одной записи, а в «шапке» которой записаны имена полей: «Фамилия И.О.», «Дата рождения», «Адрес», «Заработная плата» и т.д. Данные в этом случае могут иметь определенный тип: это может быть строка (фамилия), дата (дата рождения), число (размер заработной платы). У такой модели есть существенные ограничения, затрудняющие представление информации в виде документа. Во-первых, каждый элемент данных должен иметь фиксированную длину (не большую, чем та, которая отведена для него в таблице), из-за чего трудно работать с чрезвычайно распространенной в учрежденческих документах текстовой информацией (аннотацией на книгу в библиографической карточке, автобиографией в личном деле). Во-вторых, каждый элемент данных может встречаться в записи только один раз, в то время как некоторые пункты бумажных документов могут иметь несколько значений (например, фамилии авторов книги в библиографической карточке, телефоны организации в телефонном справочнике, начало сеанса в афише). Конечно, суще-

ствуют методы, позволяющие обходить эти ограничения и создавать базы данных достаточно сложной иерархической структуры в рамках реляционных СУБД. Не занимаясь их подробным изложением, скажем, что применение этих методов требует достаточно глубоких знаний в области программирования на специфических языках управления базами данных (dDASE, Clipper и т.п.). Именно несоответствие реляционной модели организации данных структуре учрежденческих документов делает практически неосуществимой задачу разработки ИПС для пользователя-непрофессионала в программировании. С другой стороны, никакой программист не сможет создать ИПС, не будучи специалистом в конкретной области деятельности (библиографии, бухгалтерском учете, планировании).

Не напоминает ли Вам, уважаемый читатель, сложившаяся ситуация старинную притчу о камне, лежавшем на дороге (как известно, лишь самый умный человек предпочел убрать камень с дороги, а не спотыкаться о него). Не кажется ли Вам, что действительно кардинальным решением задачи была бы разработка такой СУБД, структура данных которой полностью соответствовала бы структуре данных в учрежденческих документах. Разработка такой СУБД позволила бы разрабатывать проблемно-ориентированные ИПС специалистам в конкретной проблеме (даже в случае отсутствия у них навыков в области программирования). Именно эту задачу ставили перед собой программисты из московской фирмы «ИНФОРГ» (125047, а/я 81, Москва, ул. Тверская-Ямская, 46а-1, тел. 250-50-82), разрабатывая СУБД ИРИС (Инструмент Разработчика Информационных Систем).

СУБД ИРИС использует иерархическую (документо-. ориентированную) модель организации данных. По сравнению с реляционной моделью создатели ИРИС сделали на первый взгляд небольшую модификацию: они ввели возможность объединять несколько связанных по смыслу полей в единую группу и декларировали возможность существования множественных полей и групп. Более того, группы в ИРИС могут быть вложенными друг в друга (с сохранением возможности размножения). Все это и привело к появлению у СУБД ИРИС совершенно новых качеств. Организация данных стала многоуровневой, появилась возможность работать с текстовой информацией переменной длины, возможность объединения нескольких связанных между собой полей в группы и дальнейшей обработки их как единого целого.

Множественные поля и группы делают более полным соответствие данных компьютерных и бумажных документов. Например, все данные, относящиеся к определенному этапу научно-исследовательской работы (номер этапа, наименование работ, начало и окончание этапа, стоимость этапа), можно объединить в группу и объявить ее множественной или декларировать поле «Год издания» в ИПС «Библиотека» множественным (обратите внимание, что при этом совершенно не обязательно заранее знать ни точное количестве этапов работы, ни количество изданий книги). Введение атрибута множественности существенно упрощает работу с текстовой информацией.

Удобные диалоговые средства разработки и обслуживания делают возможной успешную работу с ИРИС даже для пользователя — непрофессионала в программировании. Специалисту для успешной работы с ИРИС нужно знание специфики своей повседневной деятельности и понимание смысла самых основных положений ИРИС. А они достаточно просты.

ИРИС — это документно-ориентированная система, работающая с базами документов. Электронные документы, используемые в ИРИС, — это заменители обычных бумажных документов, с которыми многие из нас имеет дело на работе (копаясь в библиографических карточках, например) или в быту (придя в поликлинику и толкаясь в регистратуре).

Конечно, разнообразные бумажные документы отличаются друг от друга и внешне, и содержащимися в них данными. Документы могут быть разных типов, например, «Анкета», «Накладная» и т.п. Заполненные бланки одного

типа удобнее называть экземплярами документа этого типа. Так, лежащие в папке анкеты сотрудников предприятия — это экземпляры документа типа «Анкета»; анкета Иванова является экземпляром документа типа «Анкета»; данные этого экземпляра состоят из информации об Иванове. Электронные документы в среде ИРИС характеризуются рядом «компьютерных» понятий, связанных со спецификой хранения и отображения информации в ЭВМ. Схематично электронный документ ИРИС может быть представлен состоящим из двух частей: Данных Документа (ДД) и Описания Документа (ОД).

Данные Документа составляют содержание соответствующего документа, отличающее его от других экземпляров однотипного документа. Данные представляют собой совокупность значений полей, из которых состоит документ. Каждое поле имеет тип, имя и другие атрибуты, которые в совокупности определяют описание поля. Тип поля определяет вид информации, которая может содержаться в этом поле (число, строка, дата и т.п.).

Описание Документа — это служебная информация, необходимая ИРИС для работы с документом. Все документы, имеющие одно описание, называются однотипными. Описание Документа включает описание его структуры, внешней формы и связи с Базой Документов (БД).

База Документов в ИРИС служит для хранения Данных электронных документов. Она состоит из логических файлов, каждый из которых может содержать данные документа одного типа. Каждый логический файл состоит из двух файлов DOS: собственно файла данных и файла индексов, необходимого для обеспечения быстрого поиска в БД затребованной пользователем информации.

Поясним еще одно (пожалуй, последнее) понятие, которое необходимо пользователю для успешной работы с ИРИС, — роль документа. Если бы пользователь занимался только тем, что последовательно заполнял бланки документа и помещал их на хранение базу документов — нам не потребовалось бы говорить о каких-то ролях. Но ведь заполнить новый документ и «ввести» его в машину — это только половина дела. Его еще нужно будет потом найти. Для этого удобно заполнить некий бланк «запроса» на поиск (он по своей форме немного отличается от формы искомого документа), просмотреть список найденных документов и выбрать нужный. Для выполнения этих действий в ИРИС существуют служебные документы, или, как говорят в ИРИС — документы могут играть разные роли. Всего их четыре:

- Хранимый документ используется для ввода новых документов в БД, для просмотра и корректировки документов, имеющихся в БД. Это «главная» по значимости роль документа — ведь именно по описанию хранимого документа определяется структура базы. Остальные служебные документы (документы в других ролях) всегда зависят от описания хранимого.

- Документ-Запрос служит для сообщения машине условий поиска документов в БД.

- Список. Такую роль имеет документ, предназначенный для просмотра списка найденных в БД документов.

- Меню. Документ в роли меню используется для организации выбора возможных работ. Вообще говоря, меню, в нашем понимании, вряд ли можно отнести к какому-либо документу, в ИРИС это сделано потому, что меню, как и описания всех других документов, создаются с помощью одной общей утилиты — Конструктора.

Вот, уважаемый читатель, и практически вся информация (помимо чисто технической о работе с компьютером), которая необходима для успешной работы с ИРИС. Комплекс программных средств, входящих в комплект поставки ИРИС, включает в себя ряд компонент, важнейшими из которых являются Монитор ИРИС и Конструктор Документов ИРИС, служащий для конструирования (создания описания) документов. Процесс конструирования документа заключается в заполнении специальных таблиц Конструктора, задающих структуру документа, и «рисовании» внешней формы документа, т.е. того, как он выглядит на экране. Заполнять таблицы Конструктора можно либо прямым набором на кла-

виатуре требуемой информации, либо,нажав на клавишу F4, выбором требуемого значения из списка (в том случае если информация может иметь лишь строго определенные значения). Всего имеется четыре базовых окна Конструктора, каждое из которых предназначено для ввода информации, описывающей одну из следующих характеристик документа ИРИС.

В базовом окне 1 Конструктора задаются параметры документа, которые характеризуют его в целом: роль документа, имя файла БД и некоторые другие, имеющие значение для документов в строго определенных ролях.

Для описания структуры документа служит базовое окно 2 Конструктора. Описание структуры документа состоит из определения имен полей, их типа, уровня вложенности (а, значит, принадлежности той или иной группе), атрибутов, способов обработки данных. Описание структуры заключается в заполнении специальной таблицы Конструктора. Эта таблица содержит четыре столбца, первые три из которых («Уровень», «Имя», «Тип») служат для ввода одномерных характеристик поля, а четвертый («Атрибуты») служит для отображения атрибутов, задаваемых в специальном окне.

Третье базовое окно Конструктора служит для конструирования внешней формы. Внешняя форма документа — это то, как документ выглядит на экране. Помимо полей, содержащих данные документа, внешняя форма может содержать любую пояснительную информацию. Средства, предоставляемые Конструктором для внесения пояснительной информации, аналогичны средствам любого текстового редактора. Имеется возможность рисования линий во внешней форме, включения в пояснительную информацию любого символа из таблицы ASCII. Для создания формы поля, служащего для отображения информации, нужно переместить курсор в то место формы, где должно находиться поле, и нажать F6. В дополнительном окошке, появившемся при этом, следует задать характеристики этого поля: имя, длину, опции и цвет. Интересно отметить, что имеется возможность конструирования документа без явного описания его структуры (путем заполнения таблицы Конструктора). При этом, если в процессе описания внешней формы потребуется добавить в документ новое поле, эту работу возьмет на себя Конструктор Документов (это особенно удобно в том случае, когда заранее известно как документ должен выглядеть на экране).

Четвертое базовое окно Конструктора предназначено для описания связи документа с БД. Конструктор заполняет это окно автоматически в процессе компиляции описания, и во многих случаях Вам не придется изменять информацию, содержащуюся в этом окне. Однако у Вас имеется возможность самому определить основные технические характеристики файлов БД.

Работа с БД осуществляется под управлением Монитора ИРИС и реализует основные возможности информационнопоисковой системы: ввод новых документов в БД, удаление документов из БД, сортировка найденных документов по различным полям, просмотр и редактирование найденных документов, копирование и перемещение документа или группы документов из одного файла БД в другой, печать одного или группы документов и др.

Для работы в БД используются три основных окна: окно для ввода, просмотра и корректировки документов (окно документов); окно для задания запроса на поиск (окно запроса) и окно для отображения списка найденных документов (окно списка).

Каждое из этих окон служит для отображения соответствующего документа (Хранимого документа, документа-Запроса или документа-Списка) и выполнения над ними различных операций (естественно, все необходимые документы должны быть предварительно сконструированы).

С помощью специальных команд можно переключаться с одного вида деятельности на другой, переходя в соответствующее окно. Например, можно начать ввод серии документов в БД, работая в окне документов. Затем перейти в окно поиска, сформулировать запрос на поиск, заполнив все или некоторые поля в документе-Запросе, и выполнить поиск докумен-

тов. После этого можно просмотреть список найденных документов, перейдя в окно списка, и, наконец, отредактировать какой-либо из найденных документов, вернувшись для этого в окно ввода. Отметим, что это далеко не единственный, но тем не менее весьма типичный способ работы с БД. На его примере видно, что эта работа поддерживает три основных вида деятельности: ввод новых документов, поиск документов, просмотр и редактирование найденных документов.

ИРИС — открытая для пользователя система. Для более «тонкой» настройки ИПС под Ваши нужды Вы можете использовать очень удобный механизм — модули дополнительной обработки (ДО-модули). Они применяются для организации нестандартной обработки значений полей. Можно пользоваться как стандартными ДО-модулями, входящими в комплект поставки ИРИС, так и самостоятельно создавать собственные ДО-модули. Здесь следует отметить, что для программирования наряду со средствами ИРИС Вам потребуются транслятор Си (MSC 5.0 или TC 2.0), редактор связей и текстовый редактор.

Стандартные ДО-модули, входящие в комплект поставки ИРИС, позволяют выполнять различные действия с данными некоторого поля или группы полей (проверка значения поля на принадлежность заданному диапазону, вычисление значения поля, организация ввода информации в поле из списка возможных значений). «Присоединение» ДО-модуля к полю документа происходит на этапе конструирования. Для этого в поле «Обработка» окна «Атрибуты» Конструктора следует указать имя ДО-модуля и, если это необходимо, дополнительные параметры.

Программирование в среде ИРИС имеет определенную специфику по сравнению с традиционными методами программирования, связанную в основном с динамической загрузкой программ. Динамический модуль представляет собой программный модуль, состоящий из одной или нескольких функций, некоторые из которых могут быть объявлены доступными из других программ (такие функции будем называть точками входа динамического модуля). Синтаксически динамические модули могут ничем не отличаться от обычных программ на языке Си. Специфика их состоит лишь в написании заголовков точек входа (кроме того, естественно, динамический модуль не может содержать функции main). В оперативной памяти одновременно может размещаться несколько динамических модулей, их число ограничивается только доступным размером оперативной памяти. ИРИС представляет программисту средства для динамической загрузки и выгрузки модулей.

Динамические модули выполняются под управлением Монитора ИРИС, который реализует следующие функции:

- загрузка и выгрузка модулей, причем каждый из них гарантированно присутствует в оперативной памяти в единственном экземпляре;
- передачи управления на точки входа и возврат в вызвавшую их программу;
- предоставление для всех функций общего стека для размещения параметров и переменных;
- предоставление всем динамическим модулям доступа к функциям из стандартных библиотек ИРИС и Си, размещенных в Мониторе ИРИС (такой подход позволяет существенно сократить размеры динамических модулей, так как они содержат лишь свой собственный код).

В отличие от процесса создания обычной программы на языке Си (исходный текст — компиляция — сборка) создание динамического модуля дополняется еще одним этапом — прекомпиляцией. Прекомпиляция предшествует компиляции и выполняется специальной утилитой, входящей в комплект поставки ИРИС. Прекомпилятор ИРИС формирует исходный текст программного модуля для штатных компиляторов Си.

Программирование в среде ИРИС, помимо использования стандартных типов данных и функций языка Си, предполагает использование специальных типов данных и функций, необходимых для работы с такими специфическими для ИРИС объектами, как документы, результаты поиска и др.

Основным информационным объектом в ИРИС является документ. Для работы с документом в программах необходимо использовать специальные переменные — Д-переменные. Переменные этого типа предназначены для хранения данных документа и другой связанной с ним информации. Внутренняя структура Д-переменной скрыта от программиста. Все действия над Д-переменной осуществляются с помощью специальных функций ИРИС и интерфейсной структуры типа item_ref (в дальнейшем IREF). IREF — это логический указатель на элемент Д-переменной. В ней содержатся данные, позволяющие однозначно идентифицировать этот элемент, и некоторая вспомогательная информация.

Библиотечные функции ИРИС можно разделить на пять больших групп:

- функции доступа к Д-переменным. К этой группе относятся функции создания и уничтожения Д-переменных, функции установки IREF на требуемый элемент Д-переменной, функции присваивания и получения значения элементов, функции изменения и получения характеристик элемента Д-переменной и некоторые другие функции;
- функции просмотра и редактирования документов. Эта группа включает в себя функции отображения и редактирования документов, функции выдачи сообщений, работы с клавиатурой. Функции этой группы позволяют Вам организовать работу с документами в диалоговом режиме;
- функции обслуживания Базы документов. Позволяют связывать Д-переменные с БД, позволять ввод/вывод данных из БД, проводить поиск документов в БД, осуществлять операции со списком найденных документов;
- функции поддержки динамических модулей. Эта группа включает в себя функции загрузки и выгрузки динамических модулей, функции их активизации, функции обработки событий;
- вспомогательные функции. Эта группа содержит большое число различных функций общего назначения (функции распознавания классов символов, работа со строками, функции выдачи диагностических сообщений, открытия библиотечных файлов и др.).

Специфика ДО-модулей состоит в том, что их вызовом управляет не программист, а Монитор ИРИС. Программист же должен четко представлять (и описывать), в каких ситуациях этот ДО-модуль будет вызван. Вызов ДО-модуля происходит при возникновении определенной специфической ситуации в системе (в терминологии ИРИС — события). Примерами событий могут быть, например, нажатие клавиши на клавиатуре, порождение новой Д-переменной и т.п. Для того чтобы можно было однозначно определить ситуацию, возникшую в системе, каждое событие идентифицируется своим кодом — целым числом. События, автоматически инициируемые Монитором ИРИС при выполнении тех или иных действий, составляют набор стандартных событий. Все события можно разделить на локальные и глобальные.

Локальные события — это события, относящиеся к какому- либо элементу Д-переменной. К стандартным локальным событиям относятся: нажатие клавиш, вход и выход из поля, преобразование внешнего изображения поля во внутреннее и обратно.

Глобальные события относятся ко всей Д-переменной или к какой-либо группе элементов. К стандартным глобальным событиям относятся: порождение или уничтожение Д-переменной, вставка или удаление экземпляров множественных элементов, пересчет значений полей и т.д.

Специфика программирования ДО-модулей состоит в том, что вместо обычного заголовка функции для каждой точки входа в исходном тексте задается строка следующего формата: @@ <имя точки входа> <список событий>

В момент инициализации ДО-модуля программисту известно:

- событие, приведшее к вызову ДО-модуля;
- элемент Д-переменной, для которого вызван ДО-модуль;
- блок дополнительных параметров.

Каждый ДО-модуль должен возвращать Монитору ИРИС двухбайтовый вектор, содержащий информацию, необходимую Монитору ИРИС. Общая схема реализации ДО-модулей может быть следующей.

```
@@ my_do EVENT_1 EVENT_2.........
 {
 if ( event == EVENT_1 ) {
 do_this();
 return EVENT_RC_OK;
 }
 if ( event == EVENT_2 ) {
 do that();
 return EVENT_RC_NOSTND;
 }
.................................
 }
```

Для обеспечения отладки ДО-модулей предусмотрена возможность организации трассировки вызовов точек входа.

Как видно, программирование ДО-модулей не вызывает особенного труда. Этот механизм чрезвычайно удобно использовать, так как для ДО-модулей ИРИС берет на себя наиболее ответственную и трудную часть — определение момента, когда этот модуль должен быть активизирован, Вам же остается лишь запрограммировать, что конкретно нужно сделать в этот момент.

Как было сказано выше, Монитор ИРИС, а точнее, работа FC (Find and Correkt), обеспечивает некий стандартный «сценарий» работы с ИПС. Однако информатика — такая область деятельности, которую трудно «втиснуть» в прокрустово ложе одного сценария. Рано или поздно Вам потребуется такая обработка данных, такой «сценарий» работы, который не может обеспечить ни одна из стандартных работ ИРИС. Выход из этого тупика — создание собственной работы (то есть программы, функционирующей под управлением Монитора ИРИС).

Остановимся на основных моментах, связанных с программированием работ. Основным отличием работы от ДО- модуля является то обстоятельство, что работа должна содержать программные компоненты, обеспечивающие ведение диалога с пользователем. При программировании работ программист должен соблюдать определенные соглашения. В частности, он должен «сообщить» Монитору ИРИС информацию, необходимую для правильного управления работой. Это делается путем заполнения специальной структуры заголовка Работы (TASK_CB). Часть полей этой структуры заполняется Монитором ИРИС автоматически, а за заполнение других несет ответственность программист. К числу этих полей, в частности, относятся: указатель на строку, идентифицирующую работу (task_name), указатель на блок управления работой (task_context), имя файла описания меню команд работы (task_file), номер текущей секции в меню команд (task_status), имя раздела помощи (help) и некоторые другие поля.

Выполнение работы состоит из трех этапов: инициализация, основной цикл, терминация. В ходе инициализации осуществляются необходимые подготовительные действия — порождение Д-переменных, открытие файлов, заполнение полей структуры TASK_CB.

Основной цикл — это собственно те действия, которые выполняются работой (редактирование документов, поиск, обработка и т.п.). Условие его завершения определяется программистом. Этап терминации состоит в уничтожении созданных Работой Д-переменных, освобождение выделенной памяти, закрытии открытых работой файлов и т.п.

Работе могут быть переданы параметры в таком же формате, как и функции main в программах на Си (число аргументов и массив указателей на них).

Синтаксис описания точки входа работы следующий: @# [тип результата] !<имя точки входа>(<аргументы>).

Приведенной в этой статье информации безусловно недостаточно для полного использования всех возможностей, которые предоставляет пакет ИРИС своим пользователям. Дело в том, что авторы отнюдь не ставили своей целью пересказ документации по ИРИС (к слову сказать, исчерпывающей). Целью авторов было рассказать лишь об основных моментах, выгодно отличающих его от других распространенных на отечественном рынке software СУБД. Тем читателям, которых заинтересовал этот программный продукт, мы советуем обратиться в фирму- разработчик «ИНФОРГ». Успехов Вам!

ПУБЛИЧНАЯ ЭЛЕКТРОННАЯ БИБЛИОТЕКА

Что это такое? Это богатая коллекция превосходных общедоступных программ, Public Domain и Shareware для IBM PC, полученных для распространения непосредственно от авторов и фирм-изготовителей. Вот лишь некоторые из них:

4DOS — *замечательный командный процессор, расширяющий возможности DOS.*
Automenu — *командная оболочка, облегчающая работу на компьютере.*
Con>Format — *резидентная утилита для форматирования в фоновом режиме.*
dLITE — *резидентная программа для работы с файлами формата dBASE.*
Easy Presentation Graphics — *мощный редактор деловой графики.*
Easy Project — *сетевой график управления фирмой, ресурсами, проектами.*
Egaint — *вариант игры Tetris с более сложным вращением падающих фигур.*
File Express — *простая и удобная база данных для хранения адресов и т.п.*
Flags — *учебная программа, содержащая свыше 600 флагов со всего мира.*
Galaxy Lite — *популярный текстовый процессор, очень быстрый и простой.*
GT Powercomm — *мощный коммуникационный пакет для связи через модем.*
InstaCalc — *весьма удобный резидентный процессор электронных таблиц.*
MegaCAD — *несложная в освоении система автоматического проектирования.*
MindReader — *текстовый процессор с элементами искусственного интеллекта.*
Patriquin's Utilities — *набор разнообразных сервисных программ для DOS.*
PC-Outline — *программа для составления планов, набросков, расписаний.*
PFROI — *пакет для брокеров и индивидуальных держателей ценных бумаг.*
Pianoman — *музыкальный процессор с полифоническими возможностями.*
Point & Shoot Backup/Restore — *утилита для резервного копирования.*
QEdit Advanced — *профессиональный текстовый редактор для программистов.*
SHEZ — *утилита-оболочка для работы с архивными файлами разных форматов.*
Soft87 — *эмулятор сопроцессора, повышающий точность вычислений.*
TeleDisk — *архиватор дисков, сохраняющий полную структуру каталогов.*
TSR Utilities — *набор утилит для работы с резидентными программами.*
Wynd-DOS — *прекрасная командная оболочка для DOS и управления файлами.*
WyndFields — *чрезвычайно простая и удобная в применении база данных.*

ЛЕГАЛЬНУЮ и ТОЧНУЮ
копию любой из этих программ вместе с документацией вы можете получить, став клиентом ПУБЛИЧНОЙ ЭЛЕКТРОННОЙ БИБЛИОТЕКИ

Для этого

НУЖНО ЗАКАЗАТЬ НАШ ТЕКУЩИЙ КАТАЛОГ-СПРАВОЧНИК НА ДИСКЕТЕ 5,25" 360 КБАЙТ, СОДЕРЖАЩИЙ ПОДРОБНЫЕ АННОТАЦИИ ЭТИХ И СОТЕН ДРУГИХ ДИСКОВ ПУБЛИЧНОЙ ЭЛЕКТРОННОЙ БИБЛИОТЕКИ НА РУССКОМ ЯЗЫКЕ

Милости просим в Публичную Электронную Библиотеку!

Наш адрес: *214036, Смоленск, а/я 248, МП "Такт".*
Расчетный счет: *МП "Такт" N468802 в Коммерческом банке "Днепр" г. Смоленска, МФО 258584.*
Телефон в Смоленске: *(08100) 5-58-05*

И.П.БЕРЕЗЕНЦЕВ
(г.Пермь)

Загрузка кодовых программ в КУВТ-86

Не каждый пользователь КУВТ-86 знает о том, что команду: OPEN"TT:NAME.BIN"FOR INPUT можно использовать для загрузки программ в машинных кодах с дисковода в БК-0010-01. По этой команде программа с указанным именем загрузится с адреса &O37400. После загрузки вместо команды CLOSE нужно очистить ячейку &O37356, иначе Бейсик будет работать с ошибками.

Оператор «OPEN» удобно использовать для подзагрузки из бейсиковой программы кодовых подпрограмм. К ним обращаются при помощи оператора «USR». Таким же образом можно загружать в экран рисунки. Для этого рисунок (экранную память) предварительно записывают на диск начиная с адреса &O37400.

Если оператором «OPEN» загружена кодовая (например игровая) программа, то для запуска ее надо переместить на адрес &O1000. Это делает предлагаемый фрагмент программы. Он может быть использован как самостоятельный загрузчик или вставлен в другую программу. На его основе можно сделать загрузчик с меню программ, хранящихся на диске.

Недостаток загрузчика в том, что его каждый раз надо загружать заново. Кроме того, предлагаемый загрузчик может работать только с программами, начинающимися с адреса &O1000. Адрес запуска программы можно поменять. Он находится на предпоследнем месте в блоке данных (строка 5050).

```
5000 INPUT «ИМЯ»; I$
5010 OPEN «TT:»+I$ +". BIN"FOR INPUT
5020 POKE &O37356,0%
5030 DATA &O12706, &O1000, &O12700, &O40400, &O163700
5040 DATA &O346, &O12701, &O1000, &O13702, &O350
5050 DATA &O112021, &O77202, &O104014, &O137, &O1000, 0
5060 RESTORE 5030
5070 I%=&O400
5080 DEF USR=I%
5090 READ D%
5100 IF D%TH POKEI%,D%EL5130
5110 I%=I%+2%
5120 GOTO 5090
5130 I%=USR(I%)
```

АКЦИОНЕРНОЕ ОБЩЕСТВО

Лазерные принтеры фирмы

HEWLETT PACKARD:

HP Lazer Jet IIIP, HP Lazer Jet III
Планшетные сканнеры формата А4
HEWLETT PACKARD ScanJet Plus !

Font-картриджи для лазерных принтеров HEWLETT PACKARD
с русскими фонтами TIMES COURIER
Картриджи Post Script — эмулятор языка описания графических
примитивов «Post Script» для HP LJ II совместимых принтеров

* * *

Встроенные и выносные модемы на 300/1200/2400 bps
с MNP-протоколом и без него.

* * *

Световое оборудование ArcNet u EtherNet, репиторы, активные
и пассивные усилители.

* * *

Выполнение работ по организации и инсталяции локальных
и глобальных вычислительных сетей Novell Net Ware со сдачей
под ключ.

Быстрота и качество обслуживания, широкий выбор услуг
в области современных компьютерных технологий
с гарантией качества.

В том числе ремонт и послегарантийное обслуживание:
◆IBM — совместимой компьютерной техники
(системных плат, блоков питания,
винчестеров, контроллеров
мониторов, принтеров),
◆ксероксов и телефаксов.

КРЕЙТ

КОММЕРЧЕСКИЙ ЦЕНТР:
190000
Санкт-Петербург, ул. Плеханова, 49
т. 311-13-01

Полоса подготовлена рекламным агенством А/О «Миф» (812)312-39-06; 314-78-93

М 38 **Машинные коды.** — М.: Знание, 1992. — 48 с. — (Новое в жизни, науке, технике. Сер. «Вычислительная техника и ее применение»; № 3).
ISBN 5-07-002498-1

В этом выпуске читатели познакомятся с процессом создания и загрузки программ в машинных кодах. Приведены списки команд для популярных микропроцессоров и микро-ЭВМ.
Материал рассчитан на широкий круг читателей.

2404000000 ББК 32. 973

Издание некоторых газет, журналов и даже книг может приносить выгоду.
Козьма ПРУТКОВ

Великий Козьма прозорливо поставил словечко «может». Сомневался, видимо, потому и подготовил еще такую формулу: «Если есть у тебя фонтан, заткни его; дай отдохнуть и фонтану».

Сегодня все издатели убедились, что их работа не приносит выгоды, и фонтан приходится затыкать. «Укрываться от дождя под дырявым зонтиком столь же безрассудно и глупо, как чистить зубы наждаком или сандараком». Серия брошюр «Вычислительная техника и ее применение» будет «заткнута» с пятого выпуска. Четвертый подписчики еще получат. Далее по принципу «вытапливай воск, но сохраняй мед».

Теперь выясним, «где начало того конца, которым оканчивается начало». Пока редактор ничего конкретного о судьбе выпусков ВТ сказать не может. Объясняется это просто: «Многие вещи нам непонятны не потому что наши понятия слабы; но потому что сии вещи не входят в круг наших понятий». Определенно скажу:

ДОПОДПИСКИ НЕ БУДЕТ.

Так решило руководство издательства. Пояснить логику такого решения не могу — не посвящен. Любой редактор хорошо знает: «Если хочешь быть покоен, не принимай горя и неприятностей на свой счет, но всегда отнеси их на казенный» и «козыряй!» Тем более что даже редакторам известно: «Мудрость, подобно черепаховому супу, не всякому доступна» и ясно к тому же, что «лжет непростительно, кто уверяет, будто все на свете справедливо!»

«При железных дорогах лучше сохранять двуколку». Последую и я этой мудрости и сохраню «мед», то есть все, что уже наработано, и даже приумножу.

Видимо, выпуски от 5 до 12 поступят в розничную продажу. А в конце года предполагается объявить подписку на 1993 год. Но цены...!!!

В выпуске 4 мы дадим новые сведения по спасению журнала. Так что, читатель, «огорошенный судьбою, ты все же не отчаивайся».

P.S. За текст в кавычках редактор ответственности не несет — это все Козьма Прутков.

◆ Тема следующего выпуска: ЭТИ РАЗНЫЕ БЕЙСИКИ ◆

МАШИННЫЕ КОДЫ

Редактор *Б.М.Васильев*. Мл. редактор *Н.А.Васильева*. Художник *В.Н.Конюхов*. Худож.редактор *И.А.Емельянова*. Техн.редактор *Т.В.Луговская*. Корректор *В.И.Гуляева*

ИБ № 12193

Подписано к печати 16.03.92. Формат бумаги 70x100 1/16. Бумага офсетная. Печать офсетная. Усл.печ.л.3,90. Усл.кр.-отт.8,45. Уч.-изд.л.4,60. Тираж 47373 экз. С—31. Заказ 1. Издательство «Знание». 101835, ГСП, Москва, Центр, проезд Серова, д.4. Индекс заказа 924703. Отпечатано с оригинал-макета издательства «Знание» на ордена Трудового Красного Знамени Тверском полиграфическом комбинате Министерства печати и информации Российской Федерации. 170024, г.Тверь, пр.Ленина,5.

Индекс 70195

Адрес подписчика:

Издательство «Знание»

Подписная научно-популярная серия

ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА И ЕЕ ПРИМЕНЕНИЕ

Теперь все большее число ученых склоняется к мысли, что в конечном счете появление ЭВМ сыграет для человечества не меньшую, а, по-видимому, даже большую роль, чем атомная энергия или космические полеты.

В. Т. Пушков

Цель разработчиков мощных ЭВМ "заключается в достижении трех Т (тера) — триллиона (10^{12}) операций в секунду, триллиона байт памяти и триллиона байт в секунду в линиях передачи данных.

Из статьи Э. Конкори

Наш адрес:
101835,
Москва,
Центр,
проезд
Серова, 4